VIMAL JYOTHI

ENGINEERING COLLEGE

CHEMPERI - KANNUR

LABORATORY RECORD

Name of the Student: ARJUN K JOSEPH PRN No: [illegible]

Branch: [illegible] Year/Semester: [illegible]

Name of the Laboratory: [illegible]

LABORATORY RECORD

Name AMEESHA P JOSEPH

Semester & Branch S2 . CSE-C

Roll No. 14

University Reg. No. VML21CS049

Certified that this is the bonafide record of the work done in

the EST102 Programming in C *laboratory of*

Vimal Jyothi Engineering College, Chemperi, Kannur

by

Mr. / Ms. AMEESHA P JOSEPH

Date 27/7/22

Place Chemperi

Staff in charge 27/7/22

HEAD OF THE DEPARTMENT
Dept. of Computer science & Engg.
Vimal Jyothi Engineering College
Chemperi-670 ...

Head of the Department

Internal Examiner

Signature Not Verified

Principal VJEC

INDEX

Exp. No.	Date	Name of the experiment	Page No.	Marks	Signature of the faculty
1	20-04-22	Familiarization of hardware components of a computer	2	25	
2	20-04-22	Familiarization of linux environment	6	25	
3	15-06-22	Familiarization of console I/o and operators in C	8	25	
4	22-06-22	Largest among three numbers	11	25	
5	22-06-22	Prime Number checking	13	23	27/7/22
6	22-06-22	Armstrong number checking	16	23	
7	22-06-22	Sum and average of an integer array	18	25	
8	22-06-22	Linear search	21	25	
9	22-06-22	Bubble sort	24	23	
10	18-07-22	Palindrome word checking	27	25	
11	20-07-22	String Concatenation	30	25	
12	20-07-22	Number of vowels, consonants and spaces in a string	33	25	
13	20-07-22	Sum of euclidean distance using structure	36	25	

INDEX

Exp. No.	Date	Name of the experiment	Page No	Marks	Signature of the faculty
14.	20-07-22	Employee details using structure	39	25	
15	20-07-22	Personal details using union	42	25	
16	21-07-22	Factorial of a number	45	25	
17	22-07-22	Reverse of a string	48	25	
18	22-07-22	Matrix arithmetic	51	25	23/9/22
19	22-07-22	Addition and swapping using pointers	54	25	
20	22-07-22	Sum of elements of an array using pointers	60	25	
21	22-07-22	File operations	63	25	
22	22-07-22	Number of characters, words and lines in a file	66	25	

Signature Not

Principal.V

Program 1
20-04-2022

FAMILIARIZATION OF HARDWARE COMPONENTS OF A COMPUTER

OBJECTIVE

Familiarize computer hardware components of a desktop computer (motherboard, cards, memory, slots, power cables etc..)

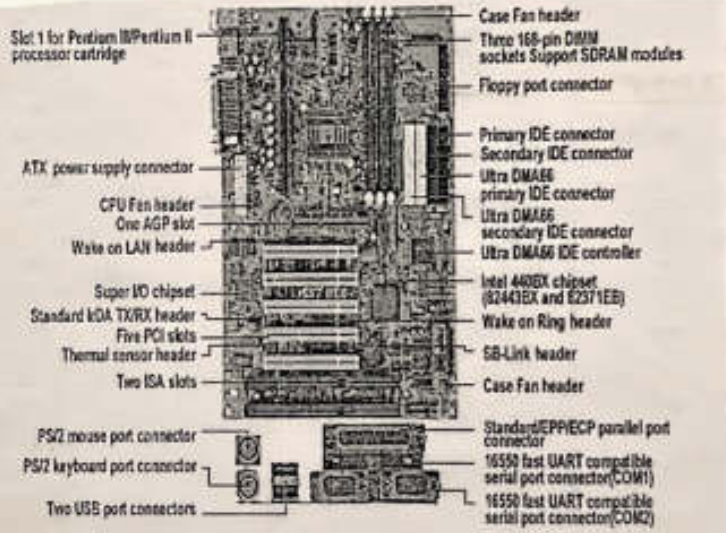
DESCRIPTION

The most important component in any PC is mother board, also called a system board. It houses a microprocessor, memory and slots for expansion of the system. Some motherboards also contain the drive interface logic, printer interface logic and serial interface logic integrated on it. Motherboards come in different sizes, shapes and ~~methods~~ models. The height and width of the motherboard is known as the motherboard form factor.
The main functional blocks of a motherboard are as follows:

Motherboard:

This is the main circuit board of the [illegible]. It contains all the basic, core components of the computer. It usually contains:

- CPU, which plugs in a socket designed for a



particular CPU's pin arrangement.

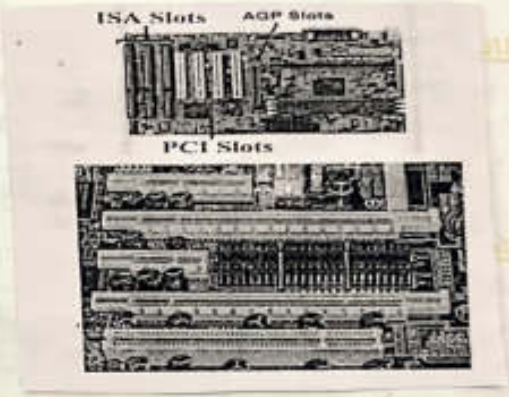
- Memory chips : These hold data and programs that the CPU is currently using.
- Input / Output ports ("I/O") such as connectors that hard disk drives, floppy disk drives and CD-ROM plugs into, serial port sockets, parallel port sockets and USB port sockets.
- BIOS chips (Basic input output system) : the BIOS chips are PROM (Programmable Read Only Memory) chips that contain the most basic information that a computer needs to startup and operate.

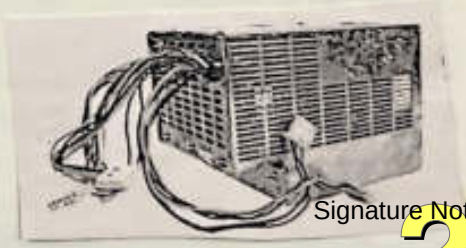
Power supply:

The power supply provides the electricity needed by the motherboard and different components in the computer. It usually provides a series of power leads carrying 12 volts or 5 volts.

Buses:

Subsystems in a PC communicate to each other via buses (basically a set of wires). Buses follow a particular set of standard rules to allow compatibility with the numerous subsystems. Every expansion card plugged into a computer's expansion slot needs to communicate with CPU.





Expansion slots:

There are sockets that expansion cards like network cards, sound cards, graphic cards can be plugged into. There have been various types of slots over the years to cater for increasingly complex expansion cards. The earliest cards were ISA, then came EISA, then PCI and AGP (which have special high priority access to the CPU-AGP is used by fast graphic cards). All expansion slots connect to a bus so that data can travel between subsystems and the CPU.

Memory:

RAM (Random Access Memory): RAM is used for storing programs temporarily. Generally, RAM is located on SIMM (Single Inline Memory Module) or DIMM (Dual Inline Memory Module).

Cache Memory: The cache is the fastest memory which lies between CPU and RAM. The CPU can access the frequently required data from cache more rapidly than from RAM.

RESULT

Familiarized with computer hardware components of a desktop computer (motherboard, cards, memory slots, power cables etc...)

Signature Not V

Principal V

Program 2
20/04/2022

FAMILIARIZATION OF LINUX ENVIRONMENT

OBJECTIVE

Familiarize basic linux commands to do programming in C with linux.

COMMANDS

clear	clear the screen
ls	listing files in the directory
ls -al	listing hidden files also
ls -l	long listing
mkdir	make a directory
pwd	present working directory
cd	change directory
touch	create a null file
gedit file	open a file in gedit text editor
cp	copy a file
mv	move a file
rm	remove a file
gcc file	compile a C program in gcc
./a.out	to execute a compiled program

RESULT

Familiarized with basic Linux commands to do programming in C with linux

Signature Not Verified

Principal VJEC

Program: 2
18-06-2022

FAMILIARIZATION OF CONSOLE I/O AND OPERATORS IN C

AIM:

Familiarize console I/O and operators in C using simple programs in C.

1. Display "Hello World"
2. Read two numbers, add them and display their sum.
3. Read the radius of a circle, calculate its area and display it.
4. Evaluate the arithmetic expression,
   $((a-b/c)*d+e)*(f+g))$
   and display its solution. Read the values of the variables from the user through console.

ALGORITHM

1. Step 1: Start
   Step 2: Print Hello World
   Step 3: Stop

2. Step 1: Start
   Step 2: Read two numbers a, b
   Step 3: Print entered numbers
   ...
   StepA: Print sum as a+b

AMEESHA P JOSEPH
VML21CS049

FAMILIARIZATION OF CONSOLE I/O AND OPERATORS IN C

1) **PROGRAM**

```
#include<stdio.h>
int main()
{
 printf("Hello World! \n");
 return 0;
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc hello.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Hello World!
```

AMEESHA P JOSEPH
VML21CS049

2) **PROGRAM**

```
#include<stdio.h>
void main()
{
int a,b;
printf("enter two numbers\n");
scanf("%d%d",&a,&b);
printf("entered numbers are %d\t%d\n",a,b);
printf("sum=%d\n",a+b);
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc sum.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
enter two numbers
5
7
entered numbers are 5 7
sum=12
```

Step 5: Stop

3. Step 1: start

Step 2: Read radius of the circle r

Step 3: Area = 3.14 * r * r

Step 4: Print area

Step 5: stop

4. Step 1: Start

Step 2: Read a, b, c, d, e, f, g

Step 3: Value = ((a - b/c * d + e)^2 (f + g))

Step 4: Print value

Step 5: Stop

RESULT:

Familiarized console I/O and operators in c using simple programs in c.

Signature Not Ver

Principal.VJ

AMEESHA P JOSEPH
VML21CS049

3) **PROGRAM**

```
#include<stdio.h>
void main()
{
int r;
float area;
printf("Enter the radius of the circle\n");
scanf("%d",&r);
area=3.14*r*r;
printf("area=%.2f\n",area);
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc area.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the radius of the circle
4
area=50.24
```

AMEESHA P JOSEPH
VML21CS049

4) **PROGRAM**

```
#include<stdio.h>
void main()
{
int a,b,c,d,e,f,g;
float value;
printf("Enter the values\n");
scanf("%d%d%d%d%d%d%d",&a,&b,&c,&d,&e,&f,&g);
value=((a-b/c*d+e)*(f+g));
printf("value=%0.2f\n",value);
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc value.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the values
10
20
30
40
50
60
70
value=7800.00
```

Program 4
Date 22/04/22

LARGEST AMONG THREE NUMBERS.

AIM:

Implement a program to find the largest among three numbers using nested if.... else statements.

ALGORITHM

Step 1: Start
Step 2: Read three numbers a, b, c
Step 3: If a>b, then go to step 4, otherwise go to step 5
Step 4: If a>c, then go to step 6, otherwise go to step 7
Step 5: If b>c, then go to step 8, otherwise go to step 7
Step 6: Largest = a, go to step 9
Step 7: Largest = c, go to step 9
Step 8: Largest = b, go to step 9
Step 9: Print Largest
Step 10: Stop

RESULT

Implemented a program to find the largest among three numbers, using nested if.... else statements.

Signature Not Verified

Principal.VJEC

LARGEST AMONG THREE NUMBERS

PROGRAM

```
#include<stdio.h>
int main()
{
  int a,b,c;
  printf("Enter three numbers\n");
  scanf("%d%d%d",&a,&b,&c);
  if(a>b)
  {
    if(a>c)
      {
        printf("largest=a\n");
      }
    else
      {
        printf("largest=c\n");
      }
  }
 else
  {
    if(b>c)
     {
        printf("largest=b\n");
     }
    else
     {
        printf("largest=c\n");
     }
  }
 return 0;
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc largest.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter three numbers
2
3
6
largest=c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter three numbers
4
3
1
largest=a
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter three numbers
6
#
3
largest=b
```

Signat

Princ

Program: 5
22/06/22

PRIME NUMBER CHECKING

AIM

Implement a program to check whether the given number is prime number or not using loop and decision making statements.

ALGORITHM

Step 1: Start
Step 2: Read a number, n
Step 3: Count = 0
Step 4: Initialize i=1
Step 5: Check whether i<=n. If it is true, then go to step 6. Otherwise go to step 9.
Step 6: If n%i == 0, then go to step 7. Otherwise go to step 8.
Step 7: Count = count + 1
Step 8: i = i+1, go to step 5
Step 9: If count == 2, then go to step 10, Otherwise, go to step 11.
Step 10: Print "Prime number" go to step 12
Step 11: Print "Not a Prime number"
Step 12: Stop

AMEESHA P JOSEPH
VML21CS049

PRIME NUMBER CHECKING

PROGRAM

```
#include<stdio.h>
int main()
 {
  int n,i,count;
  printf("enter the number\n");
  scanf("%d",&n);
  count=0;
  for(i=1;i<=n;i++)
   {
    if(n%i==0)
     {
      count=count+1;
     }
   }
    if(count==2)
     {
      printf("prime number\n");
     }
    else
     {
      printf("not a prime number\n");
     }
  return 0;
}
```

OUTPUT

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc prime.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
enter the number
3
prime number
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
enter the number
879
not a prime number
```

Signature Not Verified

Principal.VJEC

RESULT

Implemented a program to check whether the given number is prime number or not using loop and decision making statements.

Program: 6
Date 22/06/22

ARMSTRONG NUMBER CHECKING

AIM

Implement a program to check whether the given number is Armstrong or not using loop and decision making statements.

ALGORITHM

Step 1: Start
Step 2: Read a number, n
Step 3: t=n
Step 4: Initialize sum=0
Step 5: Check whether n!=0, if it is true, then go to step 6. Otherwise go to step 9.
Step 6: d= n%10
Step 7: Sum = Sum + (d*d*d)
Step 8: n = n/10, go to step 5
Step 9: If t == sum, then go to step 10. Otherwise go to step 11.
Step 10: Print "Armstrong number" go to step 12.
Step 11: Print "Not an armstrong number"
Step 12: Stop

RESULT

Implemented a program to check whether the given number is Armstrong or not using loop and decision making statements.

AMEESHA P JOSEPH
VML21CS049

ARMSTRONG NUMBER CHECKING

PROGRAM

```
#include<stdio.h>
void main()
{
 int n,t,sum,d;
 printf("Enter the number\n");
 scanf("%d",&n);
 t=n;
 sum=0;
 while(n!=0)
  {
   d=n%10;
   sum=sum+(d*d*d);
   n=n/10;
  }
 if(t==sum)
  {
   printf("Armstrong number\n");
  }
 else
  {
   printf("Not an armstrong number\n");
  }
}
```

OUTCOME

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc armstrong.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the number
153
Armstrong number
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the number
678
Not an armstrong number
```

Program 7

Date: 22-06-2022

SUM AND AVERAGE OF AN INTEGER ARRAY

AIM:

Implement a program to find the sum and average of an array of n integers.

ALGORITHM

Step 1: Start

Step 2: Read size of the array n

Step 3: Initialize i=0

Step 4: Check whether i<n, if it is true go to step 5 otherwise go to step 7

Step 5: Read a[i]

Step 6: i=i+1, go to step 4

Step 7: Print a[i]

Step 8: Initialize sum=0

Step 9: Check whether i<n. If it is true go to step 10 otherwise go to step 12

Step 10: Sum = Sum + a[i]

Step 11: i=i+1, go to step 9

Step 12: average = Sum/n

Step 13: print Sum

Step 14: Print average

Step 15: Stop

Signature Not Verified

Principal.VJEC

AMEESHA P JOSEPH
VML21CSO49

SUM AND AVERAGE OF AN ARRAY

PROGRAM

```
#include<stdio.h>
void main()
{
int a[100],n,i,sum;
float average;
printf("Enter the size of the array\n");
scanf("%d",&n);
printf("Enter the elements of the array\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
sum=0;
for(i=0;i<n;i++)
{
sum=sum+a[i];
}
average=sum/n;
printf("\nsum=%d\n",sum);
printf("average=%0.2f",average);
}
```

OUTCOME

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc array.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the size of the array
5
Enter the elements of the array
4
6
8
9
7
4       6       8       9       7
sum=34
average=6.00
```

Signature No

Principal

RESULT

Implemented a program to find the sum and average of an array of n integers.

Signature Not Verified

Principal.VJEC

Program : 8

Date 22/06/2023

LINEAR SEARCH

AIM

Implement a program to search an element in an array using linear search algorithm.

ALGORITHM

Step 1 : Start

Step 2 : Read size of the array, n

Step 3 : Initialize i=0

Step 4 : Check whether i<n. If it is true, go to step 5, otherwise go to step 7.

Step 5 : Read a[i]

Step 6 : i=i+1

Step 7 : Read element to be searched, K

Step 8 : Set j=0

Step 9 : Check whether j<n. If it is true, go to step 10. otherwise go to step 12.

Step 10 : If a[j]==K, then go to step 11. otherwise go to step 12.

Step 11 : Print "element is present at [illegible]" step 15

Step 12 : j=j+1, go to step 9

Step 13 : If j==n, then go to step 14, otherwise go to step 15

Step 14 : Print "Element is not present".

AMEESHA P JOSEPH
VML21CSO49

LINEAR SEARCH

PROGRAM

```
#include<stdio.h>
void main()
{
int a[100],i,j,k,n;
printf("Enter the size of the array\n");
scanf("%d",&n);
printf("Enter the elements of the array\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
printf("\t%d\t",a[i]);
}
printf("\nElement to be searched\n");
scanf("%d",&k);
for(j=0;j<n;j++)
{
if(a[j]==k)
{
printf("Element is present\n");
break;
}
}
if(j==n)
{
printf("Element is not present\n");
}
}
```

OUTCOME

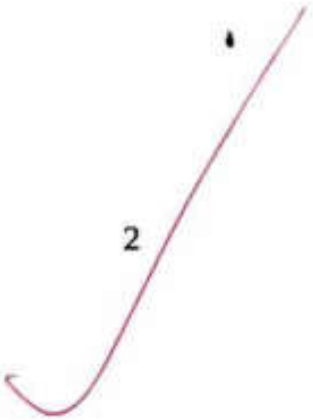
```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc linear.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the size of the array
4
Enter the elements of the array
4
6
8
2
        4               6               8               2
Element to be searched
5
Element is not present

s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc linear.c
```

Step 15 : Stop.

RESULT

Implemented a program to search an element in an array using linear search algorithm.

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the size of the array
6
Enter the elements of the array
5
4
3
2
8
8
Element to be searched
8
Element is present
```



Signature Not Verified

Principal.VJEC

Program 9

Date 22/06/2022

BUBBLE SORT

AIM:

Implement a program to sort the elements of an array using bubble sort algorithm.

ALGORITHM

Step 1: Start

Step 2: Read size of the array, n

Step 3: Set i=0

Step 4: Check whether i<n, if it is true, go to step 5, otherwise go to step 7.

Step 5: Read a[i]

Step 6: i=i+1, go to step 4

Step 7: Initialize j=1

Step 8: Check whether j<n. If it is true go to step 9, otherwise go to step 17.

Step 9: Set k=0

Step 10: Check whether k<n-j. if it is true go to step 11. otherwise go to step 16.

Step 11: If a[k] > a[k+1], then go to step 12. otherwise go to step 15.

Step 12: temp = a[k]

Step 13: a[k] = a[k+1]

Step 14: a[k+1] = temp

AMEESHA P JOSEPH
VML21CSO49

BUBBLE SORT

PROGRAM

```
#include<stdio.h>
void main()
{
int a[100],i,j,k,n,l,temp;
printf("Enter the size of the array\n");
scanf("%d",&n);
printf("Enter the elements of the array\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
for(j=1;j<n;j++)
{
for(k=0;k<n-j;k++)
{
if(a[k]>a[k+1])
{
temp=a[k];
a[k]=a[k+1];
a[k+1]=temp;
}
}
}
printf("\nSorted array is\n");
for(l=0;l<n;l++)
{
printf("%d\t",a[l]);
}
}
```

OUTCOME

```
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ gcc bubble.c
s2csec@cc21-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the size of the array
4
Enter the elements of the array
3
2
7
5
3       2       7       5
Sorted array is
2       3       5       7
```

Signature Not Verified

Principal.VJEC

step 15: k = k+1, go to step 10

Step 16: j = j+1, go to step 8

Step 17. set i = 0

Step 18: Check whether i < n, if it is true, then go to step 19. otherwise go to step 21

Step 19: Print a[i]

Step 20: i = i+1, go to step 18

Step 21: Stop

RESULT

Implemented a program to sort the elements of an array using bubble sort algorithm

21/11/22

Program No: 10

Date: 18/07/2022

PALINDROME WORD CHECKING.

AIM:

Implement a program to read a string, store it in an array and check whether it is a palindrome word or not without using built in string functions.

ALGORITHM:

Step 1: Start

Step 2: Read a string m. and store it in an array.

Step 3: Set l=0, z=0.

Step 4: Initialize i=0.

Step 5: Check whether m[i] != '\0' if it is true go to step 6, otherwise go to step 7

Step 6: l=l+1.

Step 7: i=i+1, go to step 5.

Step 8: Print length l.

Step 9: k=l

Step 10: Check whether i < k. If true go to step 11. otherwise go to step 13.

Step 11: r[i]=m[l-1]

Step 12: l=l-1.

Step 13: i=i+1

Step 14: Print reversed string

Step 15: Check whether m[i]==r[i]. If true go to step 16.

AMEESHA P JOSEPH
VML21CS049

PALINDROME WORD CHECKING

Program

```
#include<stdio.h>
int main()
{
char m[50],r[50];
int l=0,i,k,z=0;
printf("enter a string\n");
scanf("%s",m);
for(i=0;m[i]!='\0';i++)
{
l++;
}
printf("length =%d\n",l);
l=k;
for(i=0;i<k;i++)
{
r[i]=m[l-1];
l--;
}
printf("reverse:%s\n",r);
for(i=0;i<k;i++)
{
if(m[i]!=r[i])
{
z=1;
break;
}
}
if(z==0)
{
printf("palindrome\n");
}
else
{
printf("not palindrome\n");
}
return 0;
}
```

OUTPUT

```
pglab@pglab-H310M-H:~/Desktop$ gcc pal.c
pglab@pglab-H310M-H:~/Desktop$ ./a.out
enter a string
german
length =6
reverse:namreg
not palindrome
pglab@pglab-H310M-H:~/Desktop$ ./a.out
enter a string
malayalam
```

otherwise go to step 18.

Step 16: z = 1

Step 17: break.

Step 18: Check whether z == 0. If true go to step 19. otherwise go to step 20

Step 19: Print "Palindrome".

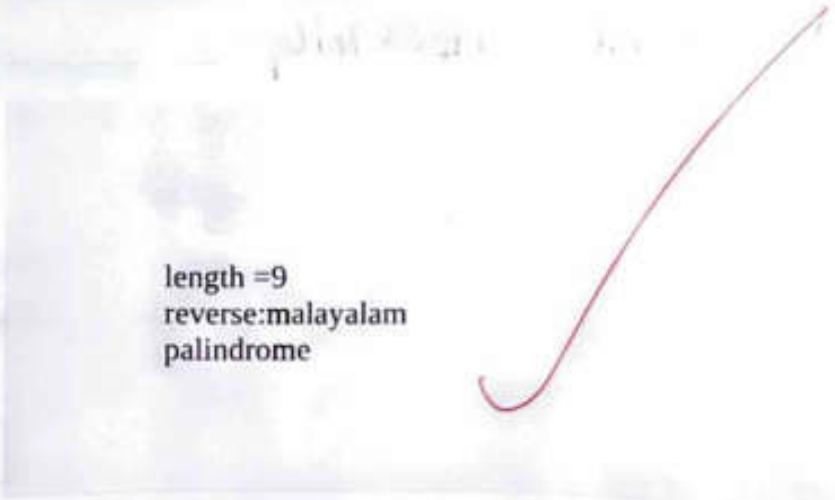
Step 20: Print "Not a palindrome".

Step 21: stop.

RESULT:

Implemented a program to read a string, store it in an array and check whether it is a palindrome word or not without using built in string functions.

22/9/22



Program 11

Date: 20/07/2022

STRING CONCATENATION

AIM:

Implement a program to read two strings (each one ending with a $ symbol), store them in arrays and concatenate them without using library function.

ALGORITHM

Step 1: Start

Step 2: Read a string and store it in a character array a of size of 100.

Step 3: Read a string and store it in a character array b of size 100.

Step 4: Initialize i=0 and n=0.

Step 5: Check whether a[i] != '\0', if its true, then go to step 6; otherwise go to step 8

Step 6: n=n+1

Step 7: i=i+1 go to step 5

Step 8: Print length of the string n.

Step 9: Initialize i=0

Step 10: Check whether b[i] != '\0' if it is true then go to step 11; otherwise go to step 14.

Step 11: Set a[n] = b[i]

Step 12: n=n+1

Step 13: i=i+1, go to step 10

AMEESHA P JOSEPH
VML21CS049

STRING CONCATENATION

PROGRAM

```
#include<stdio.h>
int main()
{
char a[100],b[100];
int n=0,I;
printf("Enter string 1\n");
scanf("%s",a);
printf("Enter string\n");
scanf("%s",b);
for(i=0;a[i]!='\0';i++)
{
n=n+1;
}
printf("The length of string is %d\n",n);
for(i=0;b[i]!='\0';i++)
{
a[n]=b[i];
n=n+1;
}
a[n]='\0';
printf("concatenated string is%s \n",a);
return 0;
}
```

OUTPUT

```
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ gcc stringcat.c
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ ./a.out
Enter string 1
hello
enter string 2
world
the length of string 1 is 5
Concatenated string is helloworld
```

Step 14 : Set a[n] = '\0'.

Step 15 : Print concatenated string, a

Step 16 : Stop.

RESULT

Implemented a program to read two strings (each one ending with a $ symbol, store them in arrays and concatenate them without using library functions

Program No:12

Date: 20.07.22

NUMBER OF VOWELS, CONSONANTS AND SPACES IN A STRING.

AIM:

Implement a program to read a string, store it in an array, and count the ~~no~~ number of vowels, consonants and spaces in it.

ALGORITHM:

Step 1 : start

Step 2 : Read a string and store in a character array ch of size 100.

Step 3 : Initialize i, ws=0, v=0, c=0, d=0, sc=0

Step 4 : Check whether ch[i] != '\0', if it is true, then go to step 5, otherwise go to step 11.

step 5 : If ch[i] = ' ', then ws = ws+1, go to step 10, otherwise go to step 6.

step 6 : If ch[i] is equal to any vowels a, e, i, o, u, then v = v+1, go to step 10, otherwise go to step 7.

Step 7 : If ch[i] is equal to any of consonants between a and z then c=c+1, go to step 10, otherwise go to step 8

Step 8 : If ch[i] is equal to any of the digits between 0 and 9. then d=d+1. go to step 10, otherwise go to step 9.

Step 9 : sc = sc+1

Step 10 : i = i+1, go to step 4.

Step 11 : print number of spaces, ws, ~~numb~~ vowels v, consonants

AMEESHA P JOSEPH
VML21CS049

NUMBER OF VOWELS,CONSONANTS,SPACES IN STRING

```
#include<stdio.h>
int main()
{
char ch[100];
int ws=0,c=0,v=0,d=0,sc=0,I;
printf("Enter the string\n);
fgets(ch,sizeof(ch),stdin);
for(i=0;ch[i]!='\0';i++)
{
if(ch[i]==' ')
{
ws++;
}
elseif(ch[i]=='a'|| ch[i]=='e'|| ch[i]=='i'|| ch[i]=='o'|| ch[i]=='u'|| ch[i]=='A'|| ch[i]=='E'||
ch[i]=='O'|| ch[i]=='I'|| ch[i]=='U')
{
v++;
}
elseif((ch[i]>='a'&&ch[i]<='z')||( ((ch[i]>='A'&&ch[i]<='Z'))
{
c++;
}
elseif((ch[i]>='0'&&ch[i]<='9')
{
d++;
}
else
{
sc++;
}
}
printf("white spaces=%d\n vowels=%d\nconsonants=%d\n digits=%d\n special characters=%d\
n",ws,v,c,d);
return 0;
}
```

OUTPUT

```
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ gcc countno.c
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ ./a.out
Enter string
Vml21CS  + ||
white spaces=3
vowels=0
consonants=5
digits=2
special characters=3
```

c, digits d and number of special characters sc.

step 12: stop

RESULT

Implemented a program to read a string, store it in an array and count the number of vowels, consonants, and spaces in it.

Program: 13

Date: 20/07/22

SUM OF EUCLIDEAN DISTANCE USING STRUCTURE

AIM

Implement a program to read two inputs each representing the distance between two points in the Euclidean space. Store this in structure variable and add two distance values.

ALGORITHM

Step 1: Start

Step 2: Create two structure variables P_1 and P_2 with data member x and y.

Step 3: Read x and y coordinates of point P_1 using structure variable P_1 as $P_1.x$ and $P_1.y$.

Step 4: Read x and y coordinates of point P_2 using structure variable P_2. $P_2.x$ and $P_2.y$

Step 5: Compute difference between x coordinate values

$P_2x - P_1.x$

Step 6: Compute the difference between y coordinate value

$P_2.y - P_1.y$.

Step 7: Calculate the distance between two points by using the operation

$d = sqrt(x*x) + (y*y)$

Step 8 : Print distance d

Step 9 : Stop

AMEESHA P JOSEPH
VML21CS049

SUM OF EUCLIDEAN DISTANCE USING STRUCTURE

PROGRAM

```
#include<stdio.h>
#include<math.h>
struct
{
int x,y;
}p1,p2;
int main()
{
int x,y;
float d;
printf("Enter x and y coordinate of point 1\n);
scanf(%d%d",&p1.x,&p1.y);
printf("Enter x and y coordinate of point 2\n);
scanf(%d%d",&p2.x,&p2.y);
x=p2.x-p1.x;
y=p2.y-p1.y;
d=sqrt((x*x)+(y*y));
printf("The distance is %0.2f\n",d);
return 0;
}
```

OUTPUT

```
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ gcc distance.c-lm
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ ./a.out
Enter x and y coordinate of point 1
10
20
Enter x and y coordinate of point 2
30
40
The distance is 28.28
```

RESULT:

Implemented a program to read two inputs each representing the distance between two points in the Eudedean space. Stored this in structure variable and added two distance values.

Program 14
Date 20/07/2022

EMPLOYEE DETAILS USING STRUCTURE

AIM:

Implement a program to read and print data of n employees (Name, employee id, salary) using structure.

ALGORITHM

Step 1: Start

Step 2: Create a structure variable e as an array of size 10 with data members as character array n of size 50, id and s

Step 3: Read the number of employee, n.

Step 4: Initialize i=0

Step 5: Check whether i<n, if it is true then go to step 6 otherwise go to step 8

Step 6: Read name, id, salary of each employee using structure variable e[i].n, e[i].id, and e[i].s

Step 7: i=i+1. go to step 5

Step 8: Initialize i=0

Step 9: Check whether i<n, if it is true go to step 10. otherwise go to step 10, otherwise go to step 11.

Step 10: Print name, id and salary of each employee using structure variable e, e[i].n, e[i].id and e[i].s

Step 11: i=i+1 go to step 9

AMEESHA P JOSEPH
VML21CS049

EMPLOYEE DETAILS USING STRUCTURE

```
#include<stdio.h>
struct
{
char n[10];
int id;
float s;
}e[10];
int main()
{
int n,I;
printf("Enter the number of employees\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter name,id and salary of Employee %d:\n",i+1);
scanf("%s%d%f",e[i].n,&e[i].id,e[i].s);
}
for(i=0;i<n;i++)
{
printf("Entered name,id and salary of employee %d: \n",i+1);
printf("%s\n%d\n%0.2f\n",e[i].n,e[i].id,e[i].s);
}
return 0;
}
```

OUTPUT

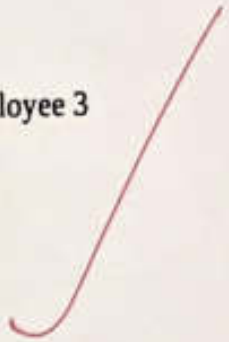
```
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ gcc employee.c
s2csec@cc106-OptiPlex-3020:~/DESKTOP/VML21CS049$ ./a.out
Enter the number of employees
3
Enter name,id and salary of Employee 1
Marcus
1
2000000
Enter name,id and salary of Employee 2
Riley
2
4000000
Enter name,id and salary of Employee 3
Eugene
3
1200000
Entered name,id and salary of Employee 1
Marcus
1
2000000.00
Entered name,id and salary of Employee 2
Riley
2
4000000.00
```

step 12 : Stop

RESULT

Implemented a program to read and print data of n employees (name, employee id, salary) using structure.

Entered name,id and salary of Employee 3
Eugene
3
1200000.00



Program 15

Date: 20/07/2022

PERSONAL DETAILS USING UNION

AIM:

Implement a program to create a union with variable name, house name, city name, state code and pin with a length of C-SIZE (User defined constants). Then display the address of person using a variable of the union

ALGORITHM

Step 1: Start

Step 2: Define a constant variable C-SIZE with value 20.

Step 3: Create a union variable t with data members as name [C-SIZE], house [C-SIZE], city [C-SIZE], state [C-SIZE] and pincode [C-SIZE].

Step 4: Read the name of the person using union variable t, t.name.

Step 5: Print the name of the person using union variable t, t.name

Step 6: Read house name of the person using union variable t, t.house

Step 7: Print the house name of the person using union variable t, t.house.

Step 8: Read name of city using union variable t, t.city

Step 9: Print name of city using union variable t, t.city

Step 10: Read the name of state using union variable

AMEESHA P JOSEPH
VML21CS049

PERSONAL DETAILS USING UNION.

Program

```
#include<stdio.h>
#include<string.h>
#define C_SIZE 20
union pdet
{
   char name[C_SIZE];
   char house[C_SIZE];
   char city[C_SIZE];
   char state[C_SIZE];
   char pincode[C_SIZE];
};
int main(){
   union pdet p;
   printf("Enter the name of the person : ");
   gets(p.name);
   printf("The name of the person is ");
   puts(p.name);
   printf("Enter the house name of the person : ");
   gets(p.house);
   printf("The house name of the person is ");
   puts(p.house);
   printf("Enter the city of the person : ");
   gets(p.city);
   printf("The city of the person is ");
   puts(p.city);
   printf("Enter the state of the person : ");
   gets(p.state);
   printf("The state of the person is ");
   puts(p.state);
   printf("Enter the pincode of the person : ");
   gets(p.pincode);
   printf("The pincode of the person is ");
   puts(p.pincode);
   return 0;
```

t, t.state

step 11: Print the name of the state using ~~structure~~ union variable t, t.state

step 12: Read the pincode using ~~struct~~ union variable t, t.pincode

Step 13: Print the pincode using union variable t, t.pincode

step 14: Stop.

RESULT.

Implemented a program to create a union with variables name, house name, city name, state code and pin with a length of C-SIZE (user defined constants). Then display the address of person using a variable of the union.

}

OUTPUT:

```
Enter the name of the person : Ameesha
The name of the person is Ameesha
Enter the house name of the person : Palukunnel
The house name of the person is Palukunnel
Enter the city of the person : Kottayam
The city of the person is Kottayam
Enter the state of the person : Kerala
The state of the person is Kerala
Enter the pincode of the person : 645671
The pincode of the person is 645671
```

Program 16
Date: 20/09/22

FACTORIAL OF A NUMBER

AIM:

Implement a program to find the factorial of a given number n using recursive and non-recursive functions.

ALGORITHM FOR MAIN FUNCTION

Step 1: Start
Step 2: Read a number, n
Step 3: Print factorial of number using recursion by calling function facts with parameter n.
Step 4: Print factorial of number using recursion by calling the function fact with parameter n.
Step 5: Stop

ALGORITHM FOR factor function - Take entered number as n.

Step 1: Start
Step 2: If n=0, then return it to the called function otherwise go to step 3.
Step 3: Return the result of m* facts (m-1) to the called function.

ALGORITHM FOR fact function - Take entered number as k.

Step 1: start

AMEESHA P JOSEPH
VML21CS049

FACTORIAL OF A NUMBER

```
#include<stdio.h>
int fact(int);
int factr(int);
int factr(int n);
 int main()
 {
 int n;
 printf("Enter a positive integer: ");
 scanf("%d",&n);
 printf("Factorial of with recursion %d = %d\n", n, factr(n));
 printf("Factorial of without recursion %d = %d", n, fact(n));
  return 0;
  }
  int factr(int m)
   { if (m>=1)
   return m*factr(m-1);
    else return 1;
    }
    int fact(int num)
    {
    int f=1,i;
    for(i=1;i<=num;i++)
     f = f * i;
     return f;
     }
```

OUTPUT

```
2csec@cc108-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter a positive integer: 25
Factorial of with recursion 25 = 2076180480
Factorial of without recursion 25 = 2076180480
```

Signature Not Verified

Principal.VJEC

Step 2: Initialize, f = 1

Step 3: If k = 0, then set f = 1, go to step 8. Otherwise go to step 4.

Step 4: Initialize i = 1

Step 5: Check whether i <= k, if it is true then go to step 6 otherwise go to step 8.

Step 6: Calculate, f = f * i

Step 7: i = i + 1, go to step 5

Step 8: Return f to the called function.

RESULT

Implemented a program to find the factorial of a given number n using recursive and non-recursive functions.

Program 17
Date 20/07/22

REVERSE OF A STRING

AIM

Implement a program to ~~react~~ reverse a string (word), store it in an array and obtain it's reverse by using user defined function.

ALGORITHM

Algorithm for main function:

step 1: Start

Step 2: Read a string and store it in a character array ch of size 100.

step 3: Initialize i=0 and n=0

step 4: Check whether ch[i]!='\0', if it is true then go to step 5, otherwise go to step 7.

Step 5: n=n+1

Step 6: i=i+1, go to step 4

Step 7: Print length of string n

step 8: Call function reverse with parameters ch and n.

Step 9: Stop

Algorithm for reverse function - Take two parameters as input, entered string as character array c and length of string m.

Step 1: Start

VML21CS049

REVERSE OF A STRING.

```
#include <stdio.h>
void strrev(char* str)
{
   char r[100];
   int a,b,c=0;
   printf("Input a string\n");
   gets(str);
   while(str[c]!='\0')
   c++;
   b=c-1;
   for(a=0;a<c;a++)
   {
     r[a]=str[b];
     b--;
   }
   r[a]='\0';
   printf("%s\n", r);
}
int main()
{
   char str[100];
   strrev(str);
   return 0;
}
```

OUTPUT:

```
s2csec@cc26-OptiPlex-3020:~/VML21CS049$ ./a.out
Input a string
germany
ynamreg
s2csec@cc26-OptiPlex-3020:~/VML21CS049$ ./a.out
Input a string
12345
54321
```

Step 2: Declare characters array rev of size 100 to store reverse of string.

Step 3: Set l = m

Step 4: Initialize i=0

Step 5: Check whether i < l, if it is true, then go to step 6 otherwise go to step 9.

Step 6 : Set rev[i] = c[m-1]

Step 7 : m = m-1

Step 8 : i = i+1, go to step 5

Step 9 : print reverse of the string, rev.

Step 10: Return to the called function.

RESULT.

Implemented a program to reverse a string (word) stored it in an array and obtained it's reverse by using user defined functions.

Program 18

Date: 21-07-22

MATRIX ARITHMETIC

AIM:

Implement a menu driven program for performing matrix addition, multiplication and finding the transpose. Use functions to,

i) read a matrix

ii) Find the sum of two matrices

iii) Find the product of two matrices

iv) Find the transpose of a matrix

v) Display a matrix

ALGORITHM

Step 1: Start

Step 2: Declare matrices mat1[10][10], mat2[10][10], tran[10][10], sum[10][10], and mult[10][10].

Step 3: Read your choice in variable c from,

1. Read the matrices
2. Sum of matrices
3. Product of matrices
4. Transpose of a matrix.

Step 4: If c=1, then go to Step 4. Otherwise go to step 10.

Step 5: Read the rows and columns ~~and~~ of first matrix, m1 and n1.

Signature Not Verified

Principal.VJEC

Step 6: Call the function ~~read~~ with parameter m1, n1

Program

```
#include<stdio.h>
#include<stdlib.h>
void displayMatrix(int,int,int [][10]);
void readMatrix(int,int,int [][10]);
void sumMatrix(int,int,int,int,int [][10],int [][10],int [][10]);
void transposeMatrix(int,int,int [][10],int [][10]);
void multiplyMatrix(int,int,int,int,int [][10],int [][10],int [][10]);
int i,j;
int main()
{
  int mat1[10][10],mat2[10][10],tran[10][10],sum[10][10], mult[10][10],m1,m2,n1,n2,c;
  while(1)
  {
    printf("1:Read the matrices \n2:Sum of matrices\n3:Product of matrices\n4:Transpose of a
matrix\nEnter your Choice: \n");
    scanf("%d",&c);
    switch(c)
    {
      case 1: printf("Enter the rows and columns of first matrix : \n");
          scanf("%d%d",&m1,&n1);
          readMatrix(m1,n1,mat1);
          printf("Enter the rows and columns of second matrix :\n");
          scanf("%d%d",&m2,&n2);
          readMatrix(m2,n2,mat2);
          break;
      case 2: sumMatrix(m1,n1,m2,n2,mat1,mat2,sum);
          break;
      case 3: multiplyMatrix(m1,n1,m2,n2,mat1,mat2,mult);
          break;
      case 4: transposeMatrix(m1,n1,mat1,tran);
          transposeMatrix(m2,n2,mat2,tran);
          break;
      default:exit(0);
          break;
    }
  }
  return 0;
}
void displayMatrix(int m,int n,int mat[][10])
{
  printf("Matrix is\n");
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      printf("%d\t",mat[i][j]);
    }
    printf("\n");
```

```
    }
}
void readMatrix(int m,int n,int mat[][10])
{
   printf("Enter the Elements\n");
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&mat[i][j]);
      }
   }
   displayMatrix(m,n,mat);
}
void sumMatrix(int m1,int n1,int m2,int n2,int mat1[][10],int mat2[][10],int sum[][10])
{
   int i,j;
   if(m1!=m2 && n1!=n2)
   {
      printf("Addition not possible\n");
   }
   else
   {
      for(i=0;i<m1;i++)
      {
         for(j=0;j<n1;j++)
         {
            sum[i][j]=mat1[i][j]+mat2[i][j];
         }
      }
      printf("The Sum matrix is \n");
      displayMatrix(m1,n1,sum);
   }
}
void transposeMatrix(int m,int n,int mat[10][10],int tran[10][10])
{
   for(i=0;i<m;i++)
   {
      for(j=0;j<n;j++)
      {
         tran[j][i]=mat[i][j];
      }
   }
   displayMatrix(n,m,tran);
}
void multiplyMatrix(int m,int n,int p,int q,int mat1[][10],int mat2[][10],int mult[][10])
{
   int i,j,k,sum=0;
   if(n!=p)
```

and mat1.

Step 7: Read the rows and columns of second matrix, m2 and n2.

Step 8: Call the function readMatrix with parameter m2, n2 and mat2.

Step 9: Break, go to step 3.

Step 10: If c=2, then go to step11. Otherwise go to step 13.

Step 11: Call the function sumMatrix with parameters m1, n1, m2, n2, mat1, mat2, and sum.

Step 12: Break, go to step 3.

Step 13: If c=3, then go to step 14. Otherwise go to step 16

Step 14: Call the function multiplyMatrix with parameters m1, n1, m2, n2, mat1, mat2 and mult.

Step 15: Break, go to step 3.

Step 16: If c=4, then go to step 17. Otherwise go to step 20.

Step 17: Call the function transposeMatrix with parameters m1, n1, mat1 and trans

Step 18: Call the function transposeMatrix with parameters m2, n2, mat2 and trans.

Step 19: Break, go to step 3.

Step 20: Call exit function with parameter, go to step 21.

Step 21: Stop.

Algorithm for displayMatrix function - Take three parameters as input, number of rows as m, number of columns as n

```
    {
      printf("The multiplication is not possible.\n");
    }
    else
    {
      for(i = 0; i < m; i++)
      {
        for(j = 0; j < q; j++)
        {
          sum = 0;
          for(k = 0; k < p; k++)
          {
            sum = sum + mat1[i][k]*mat2[k][j];
          }
          mult[i][j] = sum;
        }
      }
    }
    displayMatrix(m,q,mult);
}
```

Output

```
1:Read the matrices
2:Sum of matrices
3:Product of matrices
4:Transpose of a matrix
Enter your Choice:
1
Enter the rows and columns of first matrix :
2
2
Enter the Elements
1
2
3
4
Matrix is
1    2
3    4
Enter the rows and columns of second matrix :
2
2
Enter the Elements
5
0
6
8
Matrix is
5    0
6    8
```

```
1:Read the matrices
2:Sum of matrices
3:Product of matrices
4:Transpose of a matrix
Enter your Choice:
2
The Sum matrix is
Matrix is
6    2
9    12
1:Read the matrices
2:Sum of matrices
3:Product of matrices
4:Transpose of a matrix
Enter your Choice:
3
Matrix is
17    16
39    32
1:Read the matrices
2:Sum of matrices
3:Product of matrices
4:Transpose of a matrix
Enter your Choice:
4
Matrix is
1    3
2    4
Matrix is
5    6
0    8
1:Read the matrices
2:Sum of matrices
3:Product of matrices
4:Transpose of a matrix
Enter your Choice:
5
```

matrix as mat.

Step 1: start

Step 2: Print "Matrix is"

Step 3: Initialize i=0

Step 4: Check whether i<m, if it is true, then go to step 5, otherwise go to step 12.

Step 5: initialize j=0

step 6: Check whether j<n, if it is true, then go to step 7, otherwise go to step 11.

step 7: Print each element row by row.

step 8: Print "\t"

Step 9: j=j+1, go to step 6.

Step 10: Print "\n"

Step 11: i=i+1, go to step 4.

step 12: Return to the called function.

Algorithm for readMatrix function: - Take 3 parameters as input, number of rows as m, no. of columns as n, matrix as mat.

Step 1: Start

step 2: Read elements in to the matrix

step 3: Initialize i=0

Step 4: Check whether i<m, if it is true, then go to step 5 otherwise go to step 10

step 5: Initialize j=0

Step 6: Check whether j<n, if it is true then go to step 7, otherwise go to step 9.

Step 7: Read mat[i][j]

step 8: j=j+1, go to step 6

Step 9: i=i+1 go to step 4.

step 10: Call the functions displayMatrix with arguments m, n and mat.

step 11: Return to the called function

Algorithm for sumMatrix function:

step 1: start

step 2: If m1 != m2 and n1 != n2, then go to step 3. otherwise go to step 4

Step 3: Print "Addition is not possible" go to step 13.

step 4: Initialize i=0

step 5: Check whether i<m1, if it is true then go to step 6. otherwise go to step 11

step 6: Initialize j=0

step 7: Check whether j<n1, if it is true then go to step 8. otherwise go to step 10.

step 8: Calculate sum[i][j]=mat1[i][j]+ mat2[i][j].

Step 9: j=j+1, go to step 7

step 10: i=i+1, go to step 5

step 11: Print "The sum of the matrix"

Step 12: Call the function displayMatrix with argument r, m,

and sum

Step 13: Return to the called function.

Algorithm for transposeMatrix function:

step 1: start

step 2: Initialize i=0

step 3: Check whether i<m, if it is true then go to step 4. otherwise go to step 8

step 4: Check whether j<n, if it is true then go to step 5. otherwise go to step 7.

Step 5: set tran[j][i] = mat[i][j]

step 6: Set j=j+1, go to step 4

step 7: i=i+1, go to step 3

step 8: Call the function displayMatrix with arguments n, m and tran.

step 9: Return to the called function.

Algorithm for MultidyMatrix function.

step 1: start

step 2: If n!=p, then go to step 3. otherwise go to step 4

step 3: Print "Multiplication is not possible" go to step 17.

step 4: Initialize i=0

step 5: Check whether i<m, if its true go to step 6 otherwise step 16.

step 6: Initialize j=0.

step 7: Check whether j<q, if true go to step 8. otherwise < step 15.

Step 8: Initialize sum=0

Step 9: Initialize k=0

Step 10: Check whether k<p, if it is true then go to step 11 otherwise go to step 13.

Step 11: Calculate

Sum = sum + (mat1[i][k] * mat2[k][j])

Step 12: k=k+1, go to step 10

Step 13 - Set mat[i][j] = sum

Step 14 j=j+1, go to step 7

Step 15: i=i+1, go to step 5

Step 16: Call the function display Matrix with arguments m, q, and mult.

Step 17: Return to the called function.

RESULT

Implemented a ~~dow~~ menu driven program for performing matrix addition, multiplication, and finding the transpose. Used functions to.

i) read a matrix

ii) Find the sum of two matrices

iii) Find the product of two matrices.

iv) Find the transpose of a matrix

v) Display a matrix

Program :19
Date : 22/07/22

ADDITION AND SWAPPING USING POINTERS.

AIM:

Implement a menu driven program to do all the following using pointers

i) Add two numbers

ii) Swap two numbers

ALGORITHM

Step 1: Start

Step 2 : Read two numbers a and b

Step 3: Declare two pointer variables, p and q.

Step 4: Initialize

p = &a

q = &b

Step 5 : Read your choice in variable c form.

1. Add two numbers

2. Swap two numbers.

Step 6: If c=1, then go to step 7. Otherwise go to step 11.

Step 7: Initialize s=0.

Step 8: Compute s = *pt *q.

Step 9: Print sum, s.

Step 10: Break, go to step 19.

Step 11: If c=2, then go to step 12. Otherwise go to step 18.

Step 12: Print values of a and b before swapping *p and

AMEESHA P JOSEPH
VML21CS059

ADDITION AND SWAPPING USING POINTERS.

PROGRAM

```
#include <stdio.h>
void main()
{
   int a,b,*p,*q,sum,c,temp;
   printf("Enter two numbers: \n");
   scanf("%d%d",&a,&b);
   p=&a;
   q=&b;
   printf("1:Addition\n2:Swapping\n");
   printf("Enter the choice\n");
   scanf("%d",&c);
   switch(c)
   {
     case 1:sum=*p + *q;
            printf("Sum of the numbers = %d\n",sum);
            break;
     case 2:printf("Before swapping\nfirst = %d\nsecond = %d\n",a,b);
            temp = *q;
            *q = *p;
            *p = temp;
            printf("After swapping\nfirst = %d\nsecond = %d\n",a,b);
            break;
     default:printf("Invalid");
            break;}
}
```

OUTPUT

```
Enter two numbers:
3
7
1:Addition
2:Swapping
Enter the choice
1
Sum of the numbers = 10
```

Signature Not Verified

Principal.VJEC

*q.

Step 13: temp = *p

Step 14: *p = *q

Step 15: *q = temp

Step 16: Print values of a and b after swapping, *p and *q

Step 17: Break, go to step 19.

Step 18: Call exitfunction with parameter, go to step 19.

Step 19: Stop

RESULT

Implemented a menu driven program to do all the following using pointers.

i) Add two numbers

ii) Swap two numbers.

```
Enter two numbers:
3
6
1:Addition
2:Swapping
Enter the choice
2
Before swapping
first = 3
second = 6
After swapping
first = 6
second = 3
```

Program 20

Date: 22/02/22

SUM OF ELEMENTS OF AN ARRAY USING POINTERS

AIM:

Implement a program to find the sum of the elements stored in an array using pointers.

ALGORITHM:

Step 1: start

Step 2: Declare an array a of size 50.

Step 3: Read the number of elements, n.

step 4: Declare a pointer variable p.

step 5: Initialize p=a

step 6: Initialize i=0 and sum=0.

step 7: Check whether i<n, if its true go to step 8. Otherwise go to step 11.

step 8: Read elements to the index position i using pointer variable p, (p+i).

step 9: Compute sum= sum+*(p+i).

step 10. i=i+1, go to step 11.

step 11: print sum.

step 12: Stop.

RESULT

Implemented a program to find the sum of the elements stored in an array using pointers.

AMEESHA P JOSEPH
VML21CS049

SUM OF ELEMENTS OF AN ARRAY USING POINTER

PROGRAM

```
#include<stdio.h>
int main()
{
int a[50],i,n,sum=0,*p;
printf("Enter the number of elements : \n");
scanf("%d",&n);
printf("Enter array elements :\n ");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Array Elements are:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
p = a;
for(i=0;i<n;i++)
{
sum = sum + *p;
p++;
}
printf("\nSUM = %d\n",sum);
}
```

OUTPUT :

```
s2csec@cc108-OptiPlex-3020:~/Desktop/VML21CS049$ gcc sum89.c
s2csec@cc108-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
Enter the number of elements :
2
Enter array elements :
 1
2
Array Elements are:
1       2
SUM = 3
```

Program 21.
Date 22/07/22.

FILE OPERATIONS.

AIM:

Implement a program to create a file and perform the following operations.

i) Write data to the file
ii) Read the data in a given file and display the file content on console.
iii) Append new data and display on console.

ALGORITHM.

Step 1: Start
Step 2: Declare file pointers fp, fp1, fp2 and fp3.
Step 3: Declare a character array b of size 100.
Step 4: Call fopen function to open the file "sample.txt" in write mode and store the returned value in fp.
Step 5: Write a string to the file using fprintf function
Step 6: Close the file using fclose function with parameter fp.
Step 7: Call fopen function to open the file "sample.txt" in read mode and store the returned value in fp1.
Step 8: Repeat steps 9 to 10 until end of the file reached.
Step 9: Read word by word from file using fscanf

VML21CS049

FILE OPERATIONS

PROGRAM

```
#include<stdio.h>
int main()
{
FILE *fp1,*fp2,*fp3,*fp4;
char b[100];
char b_[100];
char c[] = "hello\n";
char d[] = "hi\n";
fp1 = fopen("sample.txt","w");
fprintf(fp1,c);
fclose(fp1);
fp2 = fopen("sample.txt","r");
fscanf(fp2,"%s",b);
printf("file content is:%s\n",b);
fclose(fp2);
fp3 = fopen("sample.txt","a");
fprintf(fp3,d);
fclose(fp3);
fp4 = fopen("sample.txt","r");
printf("total file content is:\n");
while(fscanf(fp4,"%s",b_)!=EOF)
printf("%s\n",b_);
fclose(fp4);
return 0;
}
```

OUTPUT

```
s2csec@cc108-OptiPlex-3020:~/Desktop/VML21CS049$ ./a.out
file content is:Hello
total file content is:
Hello
hi
```

function and store in the character array b.
Step 10: Print the read from the file on console, b.
Step 11: Close the file using fclose function with parameter fp1.
Step 12: Call fopen function to open the file "sample.txt" in append mode and store the returned value in fp2.
Step 13: Write a string to be appended to the file using fprintf function.
Step 14: Close the file using fclose function with parameter fp2.
Step 15: Repeat steps 16 to 17 until end of the file reached
Step 16: Read word by word from file using fscanf function and store in the character array b
Step 17: Print the read from the file on console, b.
Step 18: Close the file using fclose function with parameter fp2.
Step 19: Stop.

RESULT:
Implemented a program to create a file and performed the following operations: i) write data to the file
ii) Read the data in a given file and display the file content on console
iii) Append new data and display on console.

Program :22

Date 22/07/22

NUMBER OF CHARACTERS, WORDS AND LINES IN A FILE:

AIM:

Implement a program to open a text input file and count number of characters, words and lines in it and store the results in an output file.

ALGORITHM:

Step 1: Start

Step 2: Declare file pointers fp and fp1

Step 3: Call fopen function to open the file "First.txt" in write mode and store the returned value in fp.

Step 4: Write a string to the file using fprintf function.

Step 5: close the file using fclose function with parameter fp

step 6: Call fopen function to open the file "First.txt" in read mode and store the returned value in fp1.

Step 7: Initialize, c=0, l=0, w=0.

Step 8: Repeat step 9 to 14 until end of file is reached.

Step 9: Read character by character from file using fgetc function and store in the variable ch.

Step 10: c=ch

Step 11: If ch='\n' or ch=' ', then go to step 12. otherwise

AMEESHA P JOSEPH
VML21CS049

NUMBER OF CHARACTERS, WORDS AND LINES IN A FILE

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
FILE *f;
char c,chars=0,lines=0,words=0;
f=fopen("input.txt","w");
if(f==NULL)
{
printf("Could not open the file");
exit(0);
}
printf("Enter the text: ");
while ((c=getchar())!=EOF)
{
putc(c,f);
}
fclose(f);
f=fopen("input.txt","r");
while ((c=getc(f))!=EOF)
{
chars++;
if(c=='\n')
{
lines++;
words++;
}
else if(c==' '||c=='\t'||c=='\0')
words++;
}
printf("\nThe No of characters is %d",chars);
printf("\nThe No of lines is %d\n",lines);
printf("\nThe No of Words is %d",words);
fclose(f);
}
```

Output

```
s2csec@cc39-OptiPlex-3020:~/Desktop/vml21cs049$ gcc files2.c
s2csec@cc39-OptiPlex-3020:~/Desktop/vml21cs049$ ./a.out
Enter the text:hello world
The No of characters is 11
The No of lines is 1
The No of Words is 2
```

go to step 13.

step 12: c = c+1, go to step9.

step 13: if ch = ' ' or ch = '\t' or ch = '\n' or ch = '\0', then go to step 14. otherwise go to step 9.

step 14: w = w+1, go to step 9.

step 15: Print total number of characters c.

step 16: Print total number of lines l.

step 17: Print total number of words w.

step 18: close the file using fclose function with parameter fp1.

step 19: stop

RESULT

Implemented a program to open a test input file and count number of characters, words and lines in it and stored the results in an output file.

VIMAL JYOTHI
ENGINEERING COLLEGE
CHEMPERI, KANNUR

LABORATORY RECORD

YEAR 2022 - 2023

NAME: ADWAITH RAJESH

SEMESTER & BRANCH: S3 AOS

ROLL NO.: 04

UNIVERSITY REG. NO.: VML21AD004

Certified that this is the bonafide record of Practical work done in the Data structures Laboratory of Vimal Jyothi Engineering College, Chemperi, Kannur by Mr./ Miss Adwaith Rajesh

Date 20/12/22.

Place Chemperi

Staff-in-charge

Head of the Department

20/12/22

22-02-23

Internal Examiner

External Examiner

Signature Not Verified

Principal.VJEC

INDEX

Exp. No.	Date	Name of the Experiment	Page No.	Marks	Signature of the Faculty
1	19/9/22	Linear Search	01	45	19/9
2	19/9/22	Binary Search	03	45	19/9
3.	26/9/22	Polynomial addition	06	45	26/9
4	26/9/22	Stack operations	10	44	26/9
5	17/10/22	Queue Operations	13	44	17/10
6	29/10/22	Circular Queue Operations	16	45	29/10
7	29/10/22	Infix to postfix conversion	19	45	29/10
8	31/10/22	Linked List	23	45	31/10
9	14/11/22	Stack using Linked List	29	45	14/11/22
10	14/11/22	Queue using Linked List	33	45	14/11/22
11	28/11/22	Polynomial addition using LL	37	45	28/11/22
12	28/11/22	Binary Search Tree	42	45	28/11/22
13	17/12/22	Sorting	48	45	17/12/22
14	19/12/22	BFS traversal.	53	45	19/12/22
15	19/12/22	DFS traversal.	57	45	19/12/22

Completed 20/12/22

Signature Not Verified

Principal.VJEC

```
// linear_search.c
// a c program to implement linear search on an array of integers
// VML21AD004

#include <stdio.h>


void main() {
    int size, key, found = 0;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter %d numbers\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    printf("The array\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

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

    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            found = 1;
            printf("Key %d found at pos %d\n", key, i + 1);
        }
    }

    if (!found) {
        printf("Key does not exist in the array\n");
    }

}

/*

OUTPUT

Enter the size of the array: 5
Enter 5 numbers
1 2 3 4 5
The array
1 2 3 4 5
Enter the key to search: 3
Key 3 found at pos 3


Enter the size of the array: 5
Enter 5 numbers
1 23 56 34 23
The array
1 23 56 34 23
Enter the key to search: 5
Key does not exist in the array

*/
```

19/9/22

Linear Search

EXP:1

- AIM:

To implement a C program to read an array of integers and a key and then display indexes where the key appears.

- PSEUDOCODE:

```
1.  BEGIN
2.      OUTPUT ("Enter the size of the array")
3.      n <- USERINPUT
4.      OUTPUT("Enter the elements")
5.      a[n] <- USERINPUT
6.      OUTPUT ("Enter the key to search")
7.      Key <- USERINPUT
8.      flag <- 0

9.      FOR i=0 TO n-1
            IF a[i] == key THEN
                flag <- 1
                OUTPUT("key found at i")
            ENDIF
        ENDFOR
10.     IF flag == 0 THEN
            OUTPUT("Element does not exist")
        ENDIF
11  END.
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

- RESULT:

Output obtained and verified the output

19/9/22.

Signature Not Verified

Principal.VJEC

```c
// binary_search.c
// VML21AD004
// search for an element in an int array using binary search

#include <stdio.h>

void b_sort(int arr[], int len) {
    // sort an array using bubble sort
    for (int i = 0; i < len - 1; i++) {
        for (int j = 0; j < len - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int b_search(int arr[], int key, int low, int high) {
    // search for an element using binary search
    // returns index if found else -1
    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key) {
            return mid;
        } else if (key > arr[mid]) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

void main() {
    int size, key;

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

    int arr[size];

    printf("Enter %d numbers\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    b_sort(arr, size);
    printf("The sorted array\n");

    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    printf("Enter the key to search: ");
    scanf("%d", &key);

    int pos = b_search(arr, key, 0, size - 1);
    if (pos >= 0) {
        printf("key %d found at %d\n", key, pos + 1);
    } else {
        printf("key %d does not exist in the array\n", key);
    }
}
```

19/9/22

Binary Search.

EXP:2

AIM:

To implement a C program to read an array of integers and then search for an element in the array using Binary Search.

PSEUDOCODE:

1. BEGIN.

```
FUNCTION b_sort (arr, len)
    // sort array using Bubble sort.
    FOR i=0 TO len-2
        FOR j=0 TO len-i-2
            IF (arr[j] > arr[j+1]) THEN
                temp <— arr[j]
                arr[j] <— arr[j+1]
                arr[j+1] <— temp.
            ENDIF
        ENDFOR
    ENDFOR

FUNCTION b_search (arr, key, high, low)
    // search for a key in an array using
    // binary search.

    WHILE low <= high
        mid <— (low+high)/2
```

```
/* OUTPUT
Enter the number of elements: 5
Enter 5 numbers
5 3 2 1 4
The sorted array
1 2 3 4 5
Enter the key to search: 4
key 4 found at pos 4

Enter the number of elements: 5
Enter 5 numbers
1 2 3 4 5
The sorted array
1 2 3 4 5
Enter the key to search: 6
key 6 does not exist in the array

*/
```

```
        IF (arr[mid] == key) THEN
            RETURN mid
        ELSEIF (key > arr[mid]) THEN
            low = mid+1;
        ELSE
            high = mid-1;
        ENDIF
    ENDWHILE
    RETURN -1 // key does not exist.


OUTPUT ("Enter the size of the array")
n <- USERINPUT
OUTPUT("Enter the elements")
a[n] <- USERINPUT
OUTPUT("Enter the key to search")
key <- USERINPUT
b_sort(arr, n)
OUTPUT(arr, n)
pos <- b_search(arr, key, n-1, 0)

IF (pos >=0) THEN
    OUTPUT("Element found at pos")
ELSE
    OUTPUT("Element not found")
ENDIF
END.
```

Signature Not Verified

Principal.VJEC

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

/ /

05

RESULT:

Obtained and verified the output.

Signature Not Verified

Principal.VJEC

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

#define MAX_TERMS 100

typedef struct {
    float coef;
    int expon;
} polynomial;

polynomial terms[MAX_TERMS];

int avail = 0;

int read_poly() {
    // returns the number of terms
    int n_terms;

    printf("Enter the number of terms: ");
    scanf("%d", &n_terms);
    printf("Enter the coef and exponenets in the decreasing order of expon\n");

    for (int i = 0; i < n_terms; i++) {
        scanf("%f", &terms[avail].coef);
        scanf("%d", &terms[avail++].expon);
    }
    return n_terms;
}

void display_poly(int start, int finish) {
    if (start == finish) {
        printf("0");
        return;
    }

    for (int i = start; i < finish; i++) {
        // print with a new line if the last term
        // otherwise add a +
        printf("%.2fx^%d%s", terms[i].coef, terms[i].expon,
                        (finish - i == 1) ? "\n" : " + ");
    }
}


int compare(int a, int b) {
    /* returns
        0 if a == b
        1 if a > b
        -a if a < b
    */
    return (a == b) ? 0 : (a < b) ? -1 : 1;
}

void attach(float coef, int expon) {
    if (avail >= MAX_TERMS) {
        fprintf(stderr, "Max terms reached\n");
        exit(1);
    }

    if (!coef) return;

    terms[avail].coef = coef;
    terms[avail++].expon = expon;
}
```

26 / 9 / 22

Polynomial Addition.

EXP: 3

Aim:

To implement a C program to read two polynomials and store them in an array. Calculate the sum of the two polynomials and display the first polynomial, second polynomial and the resultant polynomial.

PSEUDOCODE

```
1- BEGIN
        MAX_TERMS ← 100

        STRUCT polynomial
                coef
                expon.

        polynomial terms[MAX_TERMS];
        avail ← 0

        FUNCTION read_poly()
                OUTPUT("Enter the number of terms")
                n_terms ← USERINPUT
                OUTPUT("Enter the coff and expon in the
                        decreasing order of expon")
                FOR i=0 TO n_terms-1
```

```
void add_poly(int starta, int finisha, int startb, int finishb,
                                        int *startd, int *finishd) {

    *startd = avail;
    while ((starta < finisha) && (startb < finishb)) {
        switch(compare(terms[starta].expon, terms[startb].expon)) {
            case 0: // a.expon == b.expon
                attach(terms[starta].coef + terms[startb].coef,
                                                terms[starta].expon);

                starta++;
                startb++;
                break;
            case 1: // a.expon > b.expon
                attach(terms[starta].coef, terms[starta].expon);
                starta++;
                break;
            case -1: // a.expon < b.expon
                attach(terms[startb].coef, terms[startb].expon);
                startb++;
                break;

        }
    }

    // add the rest of the terms of the first polynomial to the sum
    for (; starta < finisha; starta++) {
        attach(terms[starta].coef, terms[starta].expon);
    }

    // add the rest of the terms of the second polynomial to the sum
    for (; startb < finishb; startb++) {
        attach(terms[startb].coef, terms[startb].expon);
    }

    *finishd = avail;

}

void main() {
    printf("Enter the first polynomial\n");
    int t1 = read_poly();
    printf("Enter the second polynomial\n");
    int t2 = read_poly();

    int startd, finishd;

    printf("Polynomial A: ");
    display_poly(0, t1);
    printf("Polynomial B: ");
    display_poly(t1, t1 + t2);
    add_poly(0, t1, t1, t1 + t2, &startd, &finishd);
    printf("Sum (A + B) : ");
    display_poly(startd, finishd);
}

/* OUTPUT

Enter the first polynomial
Enter the number of terms: 3
Enter the coef and exponenets in the decreasing order of expon
1 3
2 1
3 0
Enter the second polynomial
Enter the number of terms: 2
Enter the coef and exponenets in the decreasing order of expon
1 4
```

```
        terms[avail].coef ← USERINPUT
        terms[avail].expon ← USERINPUT
        avail ← avail + 1
    ENDFOR
    RETURN n_terms.

FUNCTION display_poly(start, finish)
    IF start == finish THEN
        OUTPUT ("0")
    ENDIF
    FOR i = start TO finish-1
        OUTPUT (terms[i].coef terms[i].expon)
    ENDFOR

FUNCTION compare (a, b)
    IF a==b THEN
        RETURN 0
    ELSE IF (a<b) THEN
        RETURN -1
    ELSE
        RETURN 1
    ENDIF

FUNCTION attach(coef, expon)
    IF avail >= MAX_TERMS THEN
        OUTPUT("max terms reached")
        EXIT(1)
    ENDIF
```

```
1 2
Polynomial A: 1.00x^3 + 2.00x^1 + 3.00x^0
Polynomial B: 1.00x^4 + 1.00x^2
Sum (A + B) : 1.00x^4 + 1.00x^3 + 1.00x^2 + 2.00x^1 + 3.00x^0

*/
```

Signature Not Verified

Principal.VJEC

```
IF (1 coef) THEN
    RETURN
ENDIF
terms [avail].coef ← coef
terms [avail]. expon ← expon
avail ← avail + 1

FUNCTION addpoly (starta, startb, finisha, finishb, *startd
                                                * finishd)
*startd ← avail.
WHILE (starta < finisha AND startb < finishb)
    SWITCH (compare (terms [starta]. expon,
                        terms [startb]. expon))
        CASE 0 :
            attach (terms [starta]. coef +
                    terms [startb]. coef, terms [startd].
            starta ← starta + 1
            startb ← startb + 1
        CASE 1 :
            attach (terms [starta]. coef, terms [starta].
                                                expon).
            starta ← starta + 1
        CASE -1 :
            attach (terms [startb] coef ; terms [startb].
                                                expon)
            startb ← startb + 1
    ENDSWITCH
ENDWHILE
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed with in 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and complete within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

```
        FOR i = starta TO finisha - 1
            attach(terms[starta].coff, terms[starta].expon)
        ENDFOR
        FOR i = startb TO finishb - 1
            attach(terms[startb].coef, terms[startb].expon);
        ENDFOR
        *finishd = anal;

OUTPUT("Enter the first polynomial")
t1 ← read_poly()
OUTPUT("Enter the second polynomial")
t2 ← read_poly()
startd ← 0
finishd ← 0
OUTPUT("Polynomial A : ")
display_poly(0, t1)
OUTPUT("Polynomial B : ")
display_poly(t1, t1 + t2)
add_poly(0, t1, t1 + t2, &startd, &finishd).
OUTPUT("Sum (A+B) : ");
display_poly(startd, finishd);
END
```

RESULT

Obtained and verified the output.

Signature Not Verified

Principal.VJEC

```
// stack_op.c
// VML21AD004
// to implement push, pop display, on a stack

#include <stdio.h>

#define MAX 3

int stack[MAX];
int top = -1;

void push() {
    // returns the pushed value
    if (top == MAX - 1) {
        printf("Stack Overflow!! Cannot push\n");
        return;
    }
    int push_val;
    printf("Enter the value to push: ");
    scanf("%d", &push_val);
    top++;
    stack[top] = push_val;
    printf("Pushed: %d\n\n", push_val);

}

void pop() {
    if (top == -1) {
        printf("Stack Undeflow!! Cannot pop\n");
        return;

    int pop_val = stack[top];
    top--;
    printf("Popped: %d\n\n", pop_val);
}


void display() {
    int val;
    int n_top = top;
    printf("Stack: ");
    while (n_top != -1 ) {
        printf("%d ", stack[n_top]);
        n_top--;
    }
    printf("\n\n");
}


void main() {
    char ch = 'y';
    int option;
    int val;

    while (ch == 'y') {
        printf("1. Push\n2. Pop\n3. Display\n4. Quit\n");
        scanf("%d", &option);

        switch (option) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
```

Stack Operations.

Exp: 4

Aim:
To implement a C program to perform push, pop and display operations.

Output verified

PSEUDO CODE

```
1. BEGIN
        MAX_SIZE ← 3
        int stack[MAX_SIZE]
        int top ← -1
        FUNCTION push(val)
            IF top >= MAX_SIZE-1 THEN
                OUTPUT("Stack Overflow")
                RETURN
            ENDIF
            top++
            Stack[top] = val

        FUNCTION pop()
            IF top == -1 THEN
                OUTPUT("Stack underflow")
                RETURN.
            ENDIF
            OUTPUT(stack[top--])
```

```
FUNCTION display()
        FOR i = top to 0
                OUTPUT(stack[i])
        ENDFOR.
Ch ← "y"
WHILE  Ch=='y'
        OUTPUT(1. Push 2. Pop 3. Display 4. Quit)
        V ← USERINPUT

        SWITCH(v)
                CASE 1:
                OUTPUT("Enter value to push")
                val ← USERINPUT
                push(val)

                CASE 2:
                 pop()

                CASE 3:
                  display()

                CASE 4:
                  ch ← 'n'

        ENDSWITCH
ENDWHILE

END.
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						14
Total Marks (45)						44
Signature						

RESULT

Observed and verified the output.

26/7/

Signature Not Verified

Principal.VJEC

```c
// queue_op.c
// VML21AD004
// implement enqueue, dequeue, and display operation on a queue

#include <stdio.h>

#define MAX_SIZE 3

int queue[MAX_SIZE];

int front = -1;
int rear = -1;

void enqueue() {
    int val;

    if (rear == MAX_SIZE - 1) {
        printf("Queue is full\n\n");
        return;
    }

    printf("Enter the value to enqueue: ");
    scanf("%d", &val);

    if ((front == -1) && (rear == -1)) {
        front = rear = 0;
        queue[rear] = val;
        return;
    }

    queue[++rear] = val;
}

void dequeue() {
    if ((front == -1) || (front > rear)) {
        printf("Queue is empty\n\n");
        return;
    }

    if (front <= rear) {
        printf("value: %d\n", queue[front++]);
        return;
    } else {
        printf("Queue is empty\n\n");
    }

}

void display() {
    if ((front >= 0) && (rear >= 0)) {
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

void main() {
    char ch = 'y';
    while (ch == 'y') {
        int choice, val;
```

Queue Operations.

Exp:5

AIM:

To implement a C program to perform enqueue, dequeue and display operations on a queue.

PSEUDOCODE:

```
BEGIN:
    MAX_SIZE ← 3
    queue[MAX_SIZE]
    front ← -1
    rear ← -1

    FUNCTION enqueue()
        IF rear == MAX_SIZE -1 THEN
            OUTPUT("Queue is full\n\n")
            RETURN
        ENDIF
        OUTPUT("Enter the value to enqueue")
        val ← USERINPUT

        IF front == -1 && rear == -1 THEN.
            front ← 0
            rear ← 0
            queue[rear] ← val.
            RETURN.
        ENDIF
        rear ← rear+1
```

```
queue[rear] = val

FUNCTION dequeue ()
    IF (front == -1  || front > rear) THEN.
        OUTPUT ("Queue is empty").
        RETURN.
    ENDIF
    IF ( front <= rear ) THEN
        OUTPUT ("value ", queue[front])
        front ← front + 1
        RETURN.
    ENDIF
    OUTPUT ("queue is empty").

FUNCTION display ()
    IF ( front >= 0 && rear >= 0) THEN
        FOR i = 0 TO rear
            OUTPUT (queue[i])
        OUTPUT ('\n').

ch ← 'y'
WHILE (ch == 'y')
    OUTPUT ("1. Enqueue \n 2. Dequeue \n 3. Display \n
                                    4. Quit \n').
    choice ← USERINPUT.

    SWITCH( choice )
        CASE 1:
            ENQUEUE()
```

RUBRICS TO EVALUATE PROGRAMMING

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						14
Total Marks (45)						44
Signature						

Signature Not Verified

Principal.VJEC

```
CASE 2 :
    dequeue().
CASE 3 :
    OUTPUT("Queue : ")
    display()
CASE 4 :
    ch ← 'n'
DEFAULT :
    OUTPUT("Invalid Input\n\n")
END.
```

RESULT

Obtained and verified the output.

12/10/22

```c
// circular_queue.c
// VML21AD004
// implement enqueue, dequeue, and display on a circular queue

#include <stdio.h>

#define MAX_TERMS 4

int queue[MAX_TERMS];
int front = 0;
int rear = 0;

void enqueue() {
    if (front == (rear + 1) % MAX_TERMS) {
        printf("Queue Overflow\n");
        return;
    }
    int val;
    printf("Enter the value to enqueue: ");
    scanf("%d", &val);
    rear = (rear + 1) % MAX_TERMS;
    queue[rear] = val;
}

void dequeue() {
    if (front == rear) {
        printf("Queue Underflow\n");
        return;
    }
    front = (front + 1) % MAX_TERMS;
    printf("The popped element is %d\n", queue[front]);
}

void display() {
    if (front == rear) {
        printf("Queue is empty\n");
        return;
    }
    printf("Elements\n");
    int i;
    i = (front + 1) % MAX_TERMS;

    while (i != (rear + 1) % MAX_TERMS) {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX_TERMS;
    }
    printf("\n");
}

void main() {
    int choice;
    char ch = 'y';
    while (ch == 'y') {
        printf("1. Enqueue 2. Dequeue 3.Display 4.Quit\n");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
```

29/10/22. Circular Queue Operations. 16

Exp:6

Aim:

To implement circular queue operation using array.

PSEUDOCODE

```
BEGIN
    MAX_TERMS ← 4
    queue [MAX_TERMS]
    front ← 0
    rear ← 0

FUNCTION enqueue()
    IF (front == (rear + 1) % MAX_TERMS) THEN
        OUTPUT ("Queue overflow").
        RETURN.
    ENDIF
    OUTPUT ("Enter a value to enqueue");
    val ← USERINPUT
    rear ← (rear + 1) % MAX_TERMS
    queue [rear] ← val

FUNCTION dequeue()
    IF (front == rear) THEN
        OUTPUT ("Queue underflow")
        RETURN
    ENDIF
```

```
            case 4:
                ch = 'n';
                break;
            default:
                printf("Invalid Input\n");
                break;

        }
    }
}

/* OUTPUT
1. Enqueue 2. Dequeue 3.Display 4.Quit
1
Enter the value to enqueue: 10
1. Enqueue 2. Dequeue 3.Display 4.Quit
1
Enter the value to enqueue: 20
1. Enqueue 2. Dequeue 3.Display 4.Quit
1
Enter the value to enqueue: 30
1. Enqueue 2. Dequeue 3.Display 4.Quit
1
Queue Overflow
1. Enqueue 2. Dequeue 3.Display 4.Quit
2
The popped element is 10
1. Enqueue 2. Dequeue 3.Display 4.Quit
2
The popped element is 20
1. Enqueue 2. Dequeue 3.Display 4.Quit
2
The popped element is 30
1. Enqueue 2. Dequeue 3.Display 4.Quit
2
Queue Underflow
1. Enqueue 2. Dequeue 3.Display 4.Quit
1
Enter the value to enqueue: 56
1. Enqueue 2. Dequeue 3.Display 4.Quit
3
Elements
56
1. Enqueue 2. Dequeue 3.Display 4.Quit
4
1. Enqueue 2. Dequeue 3.Display 4.Quit

*/
```

```
front = (front+1) % MAX_TERMS
OUTPUT("The popped element is ", queue[front])

FUNCTION display()
    IF(front == rear) THEN
        OUTPUT("Queue is empty")
        RETURN
    ENDIF

    OUTPUT("Elements"):
    i ← (front+1) % MAX_TERMS
    WHILE(i != (rear+1) % MAX_TERMS)
        queue[i]
        i = (i+1) % MAXTERMS
    ENDWHILE

Char ch = 'y'
int choice
while(ch == 'y')
    OUTPUT("1. Enqueue 2. Dequeue 3. Display 4. Quit")
    choice ← USERINPUT

    SWITCH(choice)
        CASE 1 :
            enqueue()
        CASE 2 :
            dequeue()
        CASE 3 :
            display()
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

CASE 4 :

ch = 'n'

DEFAULT

OUTPUT ("Invalid Input")

Verified

END.

RESULT

Obtained and verified the output.

29/4/22

```c
// infix_to_postfix.c
// VML21AD004
// convert infix expression to posfix expression

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

typedef struct {
    int top;
    unsigned size;
    int *arr;
} Stack;


Stack *create_stack(unsigned size) {
    Stack *s = (Stack *)malloc(sizeof(Stack));
    s->top = -1;
    s->size = size;
    s->arr = (int *)malloc(sizeof(int) * size);
    return s;
}

int is_empty(Stack *stack ) {
    return stack->top == -1;
}

void push(Stack *stack, char c) {
    stack->arr[++stack->top] = c;
}
char pop(Stack *stack) {
    if (stack->top == -1) {
        return '$';
    }
    return stack->arr[stack->top--];
}
char peek(Stack *stack) {
    return stack->arr[stack->top];
}

int is_operand(char c) {
    return (c >= 'a' && c <= 'z') ||
           (c >= 'A' && c <= 'Z');
}

unsigned exp_length(char exp[]) {
    unsigned length = 0;
    while(exp[length] != '\0') {
        length++;
    }
    return length;
}

int prec(char c) {
    switch (c) {
        case '^':
            return 3;
        case '*':
        case '/':
            return 2;
        case '+':
        case '-':
            return 1;
    }
    return -1;
}
```

29/10/22.

Infix to Postfix Conversion.

Exp: 7.

Aim:

o/p Aim: To implement a C program to convert a given infix expression to postfix expression.

PSEUDOCODE

```
BEGIN.
    Struct Stack :
        int.top :
        int Size.
        int * arr.

    FUNCTION create_Stack (int size)
        *S ← malloc (Sizeof (Stack));
        S -> top = -1
        S -> Size = size
        S -> arr ← malloc (Sizeof (int) * Size)
        RETURN S

    FUNCTION is_empty ( Stack * Stack)
        RETURN stack -> top == -1

    FUNCTION push (Stack *Stack , char c)
        stack -> arr [++ stack -> top ] = c
```

```
void infix_to_postfix(char exp[]) {
    unsigned len = exp_length(exp);
    Stack *stack = create_stack(len);

    char new_exp[len];
    int k = 0;
    for (int i = 0; i < len; i++) {
        if (is_operand(exp[i])) {
            new_exp[k++] = exp[i];
        } else if (exp[i] == '(') {
            push(stack, exp[i]);
        } else if (exp[i] == ')') {
            while (!is_empty(stack) && peek(stack) != '(') {
                new_exp[k++] = pop(stack);
            }
            if (!is_empty(stack) && peek(stack) != '(') {
                printf("Invalid Expression\n");
                exit(1);
            }
            pop(stack); // discard the (
        } else { // operator
            // R -> L precedence
            if (!is_empty(stack) && prec(exp[i]) == 3
                        && prec(peek(stack)) == 3) {
                push(stack, exp[i]);
                continue;
            }

            if (!is_empty(stack) && (prec(exp[i]) > prec(peek(stack)))) {
                push(stack, exp[i]);
                continue;
            }

            while (!is_empty(stack) && (prec(exp[i]) <= prec(peek(stack)))) {
                new_exp[k++] = pop(stack);
            }
            push(stack, exp[i]);
        }
    }

    while (!is_empty(stack)) {
        new_exp[k++] = pop(stack);
    }

    free(stack->arr);
    free(stack);

    new_exp[k] = '\0';
    printf("%s\n", new_exp);
}

void main() {
    char exp[100];
    printf("Enter a infix expression: ");
    scanf("%s", exp);
    infix_to_postfix(exp);
}

/* OUTPUT
Enter a infix expression: a*(b+c
abc+*

Enter a infix expression: a+b^d/e
abd^e/+
*/
```

```
FUNCTION pop (*stack)
    IF stack->top == -1 THEN
        RETURN '$'
    ENDIF
    RETURN stackArr[stack->top]

FUNCTION peek (*stack)
    RETURN stack->arr[stack->top]

FUNCTION isOperand (c)
    RETURN (c>='a' && c<='z') || (c>='A' && c<='Z')

FUNCTION expLength (exp)
    length←0
    WHILE (exp[length] != '\0')
        length←length+1
    ENDWHILE
    RETURN length

FUNCTION prec (c)
    SWITCH (c)
        CASE '^':
            RETURN 3
        CASE '*':
        CASE '/':
            RETURN 2
        CASE '+':
        CASE '-':
            RETURN 1
```

```
RETURN -1

FUNCTION infix_to_postfix (exp)
    len ← exp.length(exp)
    *stack ← create_stack(len)
    new_exp[len]
    k ← 0

    FOR i=0 TO len-1
        IF (isoperand(exp[i])) THEN
            new_exp[k] = exp[i] ; k->k+1
        ELSE IF (exp[i] == '(') THEN
            push(stack, exp[i])
        ELSE IF (exp[i] == ')') THEN
            WHILE(! isempty(stack) &&
                                peek(stack) != '(')
                new_exp[k] = pop(stack)
                k ← k+1
            ENDWHILE
            IF(! isempty(stack) && peek(stack)!='(')
                OUTPUT("Invalid expression")
                EXIT(1).
            ENDIF
            pop(stack)
        ELSE
            IF(! isempty(stack) && prec(exp[i])
            -> && prec(peek(stack)) == 3 THEN
                push(stack, exp[i]).
                continue.
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Tota
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour. Experiment	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

```
        ENDIF
        IF(!isempty(stack) && (prec.[exp[i]) >
        prec.[peek(stack))) THEN
            push(stack, exp[i])
            CONTINUE
        ENDIF
        WHILE(!is_empty(stack) &&
        (prec(exp[i]) <= prec(peek(stack))
            new_exp[k] = pop(stack)
            k <- k+1
        ENDWHILE
        push(stack, exp[i])
    ENDOFOR
    WHILE (!is_empty(stack))
        new_exp[k] <- pop(stack)
        k <- k+1
    ENDWHILE
    free(stack->arr)
    free(stack).
    new_exp[k] <- '\0'
    OUTPUT(new_exp)

exp[100] <- ""
OUTPUT("Enter an infix expression")
exp <- USERINPUT
infix_to_postfix(exp).
```

RESULT

Obtained and verified the output.

29/4/22

```c
// linked_list.c
// VML21AD004
// implement linked list using structures

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

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

struct Node *head = NULL;

struct Node *create_node(int data) {
    struct Node *new_node = malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

void display_ll(struct Node *start) {
    struct Node *temp = start;

    if (!temp) {
        printf("List is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int length(struct Node *head) {
    int len = 0;
    struct Node *temp = head;
    while (temp != NULL) {
        temp = temp->next;
        len++;
    }
    return len;
}

void insert_at_pos(int pos, int data) {
    if (head == NULL && pos > 1) {
        printf("Invalid Position");
        return;
    }

    if (head == NULL) {
        head = create_node(data);
        return;
    }

    struct Node *new_node = create_node(data);

    if (head != NULL && pos == 1) {
       new_node->next = head;
       head = new_node;
       return;
    }

    struct Node *temp = head;
```

Exp: 8

31 / 10 / 22

Linked List.

23

Ip received

AIM:

To implement a C program to perform the following operation on a Linked list.

PSEUDOCODE

BEGIN

```
Struct Node:
    data.
    Struct Node *next;

head ← NULL

FUNCTION create node (data)
        new_node = malloc(sizeof(Node))
        new_node->data ← data
        new_node->next ← NULL;
        return new_node.

FUNCTION display_ll (start)
        temp = start
        IF (!temp) THEN
            OUTPUT("List is empty\n");
            RETURN
        ENDIF
```

```
    int count = 1;
    while (count < pos - 1) {
        temp = temp->next;
        count++;
    }
    new_node->next = temp->next;
    temp->next = new_node;
}

void insert_at_beg(int data) {
    insert_at_pos(1, data);
}

void insert_at_end(int data) {
    int len = length(head);
    insert_at_pos((len == 0) ? 1: len + 1, data);
}


void delete_at_pos(int pos) {
    if (head == NULL && pos > 1) {
        printf("List is empty\n");
        return;
    }

    int len = length(head);

    if (pos > len) {
        printf("Invalid Pos\n");
        return;
    }

    struct Node *temp = head;
    if (pos == 1) {
        head = head->next;
        free(temp);
        return;
    }

    int count = 1;
    struct Node *prev = head;
    while (count < pos) {
        prev = temp;
        temp = temp->next;
        count++;
    }

    prev->next = temp->next;
    free(temp);
}


void delete_at_beg() {
    delete_at_pos(1);
}


void delete_at_end() {
    int len = length(head);
    delete_at_pos((le  == 0) ? 1 : len);
}

void free_ll(struct Node *head) {
    struct Node *p = head
    while (p != NULL) {
        p = p->next;
```

```
WHILE (temp != NULL)
        OUTPUT (temp->data)
        temp ← temp->next.
ENDWHILE

FUNCTION length (*head) {
    len = 0
    *temp = head
    WHILE (temp != NULL)
        temp ← temp->next.
        len ← len + 1
    ENDWHILE

FUNCTION insert_at_pos (pos, data)
    IF (head == NULL && pos > 1) THEN
        OUTPUT ("Invalid Position")
        RETURN
    ENDIF
    IF (head == NULL) THEN
        head ← create_node (data)
        RETURN
    ENDIF
    new_node ← create node(data)

    If (head != NULL && pos == 1) THEN
        new_node->next ← head
        head ← new_node
        RETURN
    ENDIF
```

```
        free(p);
    }
}


int main() {
    char ch = 'y';
    int choice, pos, val;
    while(ch == 'y') {
        printf("1. Insert at Beg. 2. Insert at End 3. Insert at pos\n");
        printf("4. Delete at Beg. 5. Delete at End 6. Delete at pos\n");
        printf("7. Display 8. Quit\n");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
            case 2:
               printf("Enter the value: ");
               scanf("%d", &val);

               if (choice == 1) {
                    insert_at_beg(val);
               } else {
                    insert_at_end(val);
               }
               break;
            case 4:
                delete_at_beg();
                break;
            case 5:
                delete_at_end();
                break;
            case 3:
            case 6:
                printf("Enter the position: ");
                scanf("%d", &pos);

                if (choice == 3) {
                    printf("Enter the value: ");
                    scanf("%d", &val);
                    insert_at_pos(pos, val);
                } else {
                    delete_at_pos(pos);
                }
                break;
            case 7:
                display_ll(head);
                break;
            case 8:
                ch = 'n';
                break;
            default:
                printf("Invalid Choice\n");
                break;
        }
        printf("\n");

    }

    free_ll(head);
    return 0;
}

/* OUTPUT
1. Insert at Beg. 2. Insert at End 3. Insert at pos
```

```
temp ← head
count ← 1;
WHILE (count < pos-1)
    temp ← temp -> next
    count ← count+1
ENDWHILE
new_node -> next ← temp->next
temp->next ← new_node

FUNCTION insert_at_beg (data)
    insert_at_pos(1, data);

FUNCTION insert_at_end(data)
    len ← length(head)
    insert_at_pos (len+1, data)

FUNCTION delete_at_pos (pos)
    IF (head == NULL && pos > 1)
        OUTPUT ("List is empty")
        RETURN
    ENDIF
    len ← length(head)
    IF( pos > len)
        OUTPUT ("Invalid Pos\n")
        RETURN
    ENDIF
    *temp ← head
    IF (pos == 1)
        head ← head -> next.
```

```
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
1
Enter the value: 10

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
1
Enter the value: 20

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
7
20 10

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
2
Enter the value: 30

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
7
20 10 30

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
3
Enter the position: 2
Enter the value: 22

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
7
20 22 10 30

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
6
Enter the position: 3

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
7
20 22 30

1. Insert at Beg. 2. Insert at End 3. Insert at pos
4. Delete at Beg. 5. Delete at End 6. Delete at pos
7. Display 8. Quit
8


*/
```

```
        free(temp)
        RETURN
    ENDIF
    count ← 1
    *prev = head;
    WHILE (count < pos)
        prev = temp
        temp ← temp->next
        count ← count + 1
    ENDWHILE
    prev->next ← temp->next
    free (temp)

FUNCTION delete_at_beg()
    delete_at_pos(1);

FUNCTION delete_at_end()
    len ← length(head)
    delete_at_pos ((len==0)?1:len)

FUNCTION free_ll(*head)
    *p ← head
    WHILE (P!=NULL)
        p ← p->next
        free(p)
    ENDWHILE
```

```
FUNCTION main()
    ch ← 'y'
    choice ← pos ← val ← 0

    WHILE(ch == 'y')

        OUTPUT("1. Insert at Beg 2. Insert at End
                3. Insert at pos\n")
        OUTPUT("4. Delete at Beg 5. Delete at End.
                6. Delete at pos\n")
        SWITCH(choice)
            CASE 1:
            CASE 2:
                OUTPUT("Enter the value:")
                val ← USERINPUT
                IF(choice == 1) THEN
                    insert_at_beg(val)
                ELSE
                    insert_at_end(val)
                ENDIF
                break
            CASE 4:
                delete_at_beg()
                break
            CASE 5:
                delete_at_end()
                break
            CASE 6:
            CASE 3:
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

```
                OUTPUT("Enter the position:")
                pos <- USERINPUT
                IF(choice == 3)
                    OUTPUT("Enter the value:")
                    val <- USERINPUT
                    insert_at_pos(pos, val)
                ELSE
                    delete_at_pos(pos)
                ENDIF
                break
            CASE 7:
                display_ll(head).
                break
            CASE 8:
                ch <- 'n'
                break
            DEFAULT:
                OUTPUT("Invalid choice\n"):
                break.
            ENDSWITCH.
        ENDWHILE
        free_ll(head):
END.
```

RESULT

— Obtained and verified the output.

```c
// stack_ll.c
// VML21AD004
// implement stack using linked list

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


typedef struct item {
    int data;
    struct item *next;
} Item;


Item *stack = NULL;

Item *create_node(int data) {
    Item *new_node = malloc(sizeof(Item));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

void display() {
    if (stack == NULL) {
        printf("Stack is empty\n");
        return;
    }

    Item *tmp = stack;
    while(tmp != NULL) {
        printf("%d ", tmp->data);
        tmp = tmp->next;
    }
    printf("\n");
}


void push(int data) {
    Item *new_node = create_node(data);

    if (stack == NULL) {
        stack = new_node;
        return;
    }

    Item *tmp = stack;
    while (tmp->next != NULL) {
        tmp = tmp->next;
    }
    tmp->next = new_node;
}


void pop() {
    if (stack == NULL) {
        printf("stack Underflow\n");
        return;
    }

    if (stack->next == NULL) {
        printf("Popped: %d\n", stack->data);
        free(stack);
        stack=NULL;
        return;
    }
```

14 / 11 / 21

Stack Using Linked List.

Expt:1

AIM:

To implement the stack datastructure using linked listed list and perform stack operations.

PSEUDOCODE

```
BEGIN.
    Struct Item
        data.
        next.

    Stack ← NULL

    FUNCTION create_node (data)
        new_node ← malloc(sizeof(Item)).
        new_node->data ← data.
        new_node->next ← NULL.
        RETURN new_node.

    FUNCTION display()
        if (stack == NULL) THEN
            OUTPUT ("stack is empty")
            RETURN.
        ENDIF
        tmp ← Stack.
```

```
    Item *tmp = stack;
    Item *prev;
    while (tmp->next!= NULL) {
        prev = tmp;
        tmp = tmp->next;
    }
    printf("Popped: %d\n", tmp->data);
    free(tmp);
    prev->next = NULL;
}

void free_stack() {
    Item *p = stack;
    while (p != NULL) {
        p = p->next;
        free(p);
    }
}

int main() {
    int ch = 1;
    int choice, val;

    while (ch) {
        printf("1. Push 2. Pop 3. Display 4. Quit\n");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                ch = 0;
                break;
            default:
                printf("Invalid Input");
                break;
        }
    }
    free_stack();
    return 0;
}

/* OUTPUT
1. Push 2. Pop 3. Display 4. Quit
1
Enter the value to push: 10
1. Push 2. Pop 3. Display 4. Quit
1
Enter the value to push: 20
1. Push 2. Pop 3. Display 4. Quit
1
Enter the value to push: 30
1. Push 2. Pop 3. Display 4. Quit
3
10 20 30
```

```
tmp ← Stack.
WHILE(tmp != NULL).
    OUTPUT(tmp->data).
    temp ← tmp->next.
ENDWHILE

FUNCTION push(data).
    new_node ← create_node(data).
    IF (Stack == NULL) THEN
        Stack ← new_node
        RETURN.
    ENDIF
    tmp ← Stack.
    WHILE (tmp->next != NULL)
        tmp ← tmp->next.
    ENDWHILE
    tmp->next ← new_node.

FUNCTION pop()
    IF (Stack == NULL) THEN.
        OUTPUT("Stack Underflow")
        RETURN.
    ENDIF
    IF (Stack->next == NULL) THEN.
        OUTPUT("Popped ", Stack->data)
        free(Stack)
        Stack ← NULL
        RETURN.
    ENDIF
```

```
1. Push 2. Pop 3. Display 4. Quit
2
Popped: 30
1. Push 2. Pop 3. Display 4. Quit
2
Popped: 20
1. Push 2. Pop 3. Display 4. Quit
2
Popped: 10
1. Push 2. Pop 3. Display 4. Quit
2
stack Underflow
1. Push 2. Pop 3. Display 4. Quit
3
Stack is empty
1. Push 2. Pop 3. Display 4. Quit
4

*/
```

```
tmp ← stack.
prev ← NULL
WHILE (tmp->next != NULL)
    prev ← tmp
    tmp ← tmp->next.
ENDWHILE
OUTPUT ("Poped ", tmp->data)
free(tmp).
prev->next = NULL.

FUNCTION free_stack()
    p ← stack
    WHILE (p != NULL)
        p ← p->next.
        free(p)
    ENDWHILE.

ch ← 1
Choice ← 0
val ← 0.

WHILE (ch)
    OUTPUT ("1. Push 2. Pop 3. Display 4. Quit")
    Choice ← USERINPUT

    SWITCH (choice)
        Case 1 :
            prompt("Enter the value to push :");
            val ← USERINPUT
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour. Start	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

```
            push(val).
        Case 2 :
            pop()
        case 3 :
            display()
        case 4 :
            ch = 0
        default :
            OUTPUT("Invalid Input").

    free_stack()
```

RESULT.

Obtained and verified the output.

14/11/25

```c
// queue_ll.c
// VML21AD004
// implement a queue using linked list


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


typedef struct item {
    int data;
    struct item *next;
} Item;


Item *queue = NULL;

Item *create_node(int data) {
    Item *new_node = malloc(sizeof(Item));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

void display() {
    if (queue == NULL) {
        printf("Queue is empty\n");
        return;
    }

    Item *tmp = queue;
    while(tmp != NULL) {
        printf("%d ", tmp->data);
        tmp = tmp->next;
    }
    printf("\n");
}


void enqueue(int data) {
    Item *new_node = create_node(data);

    if (queue == NULL) {
        queue = new_node;
        return;
    }

    Item *tmp = queue;
    while (tmp->next != NULL) {
        tmp = tmp->next;
    }
    tmp->next = new_node;
}


void dequeue() {
    if (queue == NULL) {
        printf("queue Underflow\n");
        return;
    }

    Item *tmp = queue;
    printf("Popped: %d\n", tmp->data);
    queue = queue->next;
    free(tmp);
}
```

14 / 11 / 22

Queue Using Linked list.

Expt:10

AIM.

To implement the Queue Data structure using Linked List and perform queue operations.

PSEUDOCODE

BEGIN.

```
Struct Item
        data.
        next.

Queue ← NULL

FUNCTION create_node(data).
        new_node ← malloc(sizeof(Item)).
        new_node -> data ← data.
        new_node->next ← NULL.
        RETURN new_node.

FUNCTION display().
        IF (queue == NULL) THEN.
                OUTPUT("Queue is empty").
                RETURN.
        ENDIF.
        tmp ← queue.
```

```
void free_queue() {
    Item *p = queue;
    while (p != NULL) {
        p = p->next;
        free(p);
    }
}

int main() {
    int ch = 1;
    int choice, val;

    while (ch) {
        printf("1. Enqueue 2. Dequeue 3. Display 4. Quit\n");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &val);
                enqueue(val);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                ch = 0;
                break;
            default:
                printf("Invalid Input");
                break;
        }
    }
    free_queue();
    return 0;
}

/* OUTPUT
1. Enqueue 2. Dequeue 3. Display 4. Quit
1
Enter the value to enqueue: 10
1. Enqueue 2. Dequeue 3. Display 4. Quit
1
Enter the value to enqueue: 20
1. Enqueue 2. Dequeue 3. Display 4. Quit
1
Enter the value to enqueue: 30
1. Enqueue 2. Dequeue 3. Display 4. Quit
3
10 20 30
1. Enqueue 2. Dequeue 3. Display 4. Quit
2
Popped: 10
1. Enqueue 2. Dequeue 3. Display 4. Quit
2
Popped: 20
1. Enqueue 2. Dequeue 3. Display 4. Quit
2
Popped: 30
1. Enqueue 2. Dequeue 3. Display 4. Quit
2
```

```
WHILE (tmp != NULL)
    OUTPUT (tmp -> data).
    tmp ← tmp -> next.
ENDWHILE.

FUNCTION enqueue (data)
    new_node ← create_node (data)
    IF (queue == NULL) THEN.
        queue ← new_node.
        RETURN
    ENDIF.
    tmp ← queue.
    WHILE (tmp -> next != NULL)
        tmp ← tmp -> next.
    ENDWHILE.
    tmp -> next ← new_node.

FUNCTION dequeue ()
    IF (queue == NULL) THEN.
        OUTPUT ("queue underflow")
        RETURN.
    ENDIF.
    tmp ← queue.
    OUTPUT ("Popped ", tmp -> data)
    queue <- queue -> next
    free (tmp).
```

```
queue Underflow
1. Enqueue 2. Dequeue 3. Display 4. Quit
3
Queue is empty
1. Enqueue 2. Dequeue 3. Display 4. Quit
4
*/
```

```
FUNCTION freequeue()
        p ← queue
        WHILE (p!=NULL)
            p ← p->next
            free(p).
        ENDWHILE.

ch ← 1
choice ← 0
val ← 0.
WHILE(ch)
        OUTPUT ("1. Enqueue 2. Dequeue 3. Display
                4. Quit").
        Choice ← USERINPUT.

        SWITCH (choice)
            CASE1 :
                OUTPUT ("Enter the value to enqueue")
                val ← USERINPUT
                enqueue(val).
                break
            CASE 2 :
                dequeue()
            CASE 3 :
                display()
            CASE4 :
                ch ← 0
            DEFAULT
                output("Invalid input").
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						46
Signature						

Signature Not Verified

Principal.VJEC

RESULT

Obtained and verified the output.

14/11/22

```
// poly_ll.c
// VML21AD004
// implement polynomial addition using linked list


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


typedef struct term {
    float coef;
    int exp;
    struct term *next;
} Term;

typedef struct poly {
    Term *terms;
    Term *curr;
} Polynomial;


Term *create_term(float coef, int exp) {
    Term *new_term = malloc(sizeof(Term));
    new_term->coef = coef;
    new_term->exp = exp;
    new_term->next = NULL;
    return new_term;
}

Polynomial *create_poly() {
    Polynomial *new_poly = malloc(sizeof(Polynomial));
    new_poly->terms = NULL;
    new_poly->curr = NULL;
    return new_poly;
}

void display_poly(Polynomial *p) {
    Term *tmp = p->terms;
    while(tmp != NULL) {
        printf("%.2fx^%d%s", tmp->coef, tmp->exp,
                    (tmp->next == NULL) ? "\n" : " + ");
        tmp = tmp->next;
    }
}

void attach(Polynomial *p, Term *t) {
    if (!(t->coef)) return;

    if (p->terms == NULL) {
        p->terms = t;
        p->curr = t;
        return;
    }
    p->curr->next = t;
    p->curr = t;
}

Polynomial *read_poly() {
    float coef;
    int exp;
    int n_terms;
    Polynomial *new_poly = create_poly();

    printf("Enter the number of terms: ");
    scanf("%d", &n_terms);
```

28/11/22

Polynomial Addition Using Linked List.

37

Exp:1

Aim:

To add two given polynomials using their linked list representation.

PSEUDOCODE

BEGIN

```
Struct Term
    coef.
    exp.
    next.
Struct Polynomial.
    terms.
    ener.

FUNCTION create_term(coef, exp)
    new_term ← malloc(sizeof(Term))
    new_term -> coef ← coef.
    new_term -> exp ← exp.
    new_term -> next ← NULL;
    RETURN new_term.

FUNCTION display_poly (p)
    tmp ← p -> terms.
    WHILE (tmp != NULL)
        OUTPUT(tmp->coef, tmp->exp)
        tmp ← tmp->next.
```

```
    printf("Enter the terms, decreasing order of exp\n");
    for (int i = 0; i < n_terms; i++) {
        scanf("%f", &coef);
        scanf("%d", &exp);
        attach(new_poly, create_term(coef, exp));
    }
    new_poly->curr = new_poly->terms;
    return new_poly;
}

int compare(int a, int b) {
    return (a == b) ? 0 : (a < b) ? -1 : 1;
}

Polynomial *add_poly(Polynomial *p1, Polynomial *p2) {
    Polynomial *sum = create_poly();

    while (p1->curr != NULL && p2->curr != NULL) {
        switch (compare(p1->curr->exp, p2->curr->exp)) {
            case 0:
                attach(sum, create_term(
                    p1->curr->coef + p2->curr->coef,
                    p1->curr->exp
                ));
                p1->curr = p1->curr->next;
                p2->curr = p2->curr->next;
                break;
            case 1:
                attach(sum, create_term(p1->curr->coef, p1->curr->exp));
                p1->curr = p1->curr->next;
                break;
            case -1:
                attach(sum, create_term(p2->curr->coef, p2->curr->exp));
                p2->curr = p2->curr->next;
                break;
        }
    }

    while (p1->curr != NULL) {
        attach(sum, create_term(p1->curr->coef, p1->curr->exp));
        p1->curr = p1->curr->next;
    }

    while (p2->curr != NULL) {
        attach(sum, create_term(p2->curr->coef, p2->curr->exp));
        p2->curr = p2->curr->next;
    }
    sum->curr = sum->terms;
    return sum;

}


void free_poly(Polynomial *p) {
    Term *tmp = p->terms;
    Term *ntmp;

    while(tmp != NULL) {
        ntmp = tmp;
        tmp = tmp->next;
        free(ntmp);
    }
    free(p);
}
```

Signature Not Verified

Principal.VJEC

```
FUNCTION attach (p, t).
    IF (! t -> coef) THEN
        RETURN.
    ENDIF.
    IF (p -> terms == NULL) THEN
        p -> terms <- t.
        p -> curr <- t.
        RETURN.
    ENDIF.
    p -> curr -> next <- t
    p -> curr <- t

FUNCTION read_poly ()
    newpoly <- create_poly().
    OUTPUT("Enter the numbers of terms").
    n_terms <- USERINPUT
    OUTPUT("Enter the terms").
    FOR (i = 0 TO n_terms - 1)
        coef <- USERINPUT.
        exp <- USERINPUT
        attach (new_poly, create_term (
                                coef, exp))
    ENDFOR
    new_poly -> curr <- new_poly -> terms
    RETURN new_poly

FUNCTION compare (a, b).
    RETURN (a==b) ? 0 : (a<b) ? -1 : 1;
```

```
int main() {
    printf("Enter Poly 1\n");
    Polynomial *p1 = read_poly();

    printf("Enter Poly 2\n");
    Polynomial *p2 = read_poly();

    Polynomial *sum = add_poly(p1, p2);

    printf("Polynomial A: ");
    display_poly(p1);

    printf("Polynomial B: ");
    display_poly(p2);

    printf("Sum (A + B): ");
    display_poly(sum);

    free_poly(p1);
    free_poly(p2);
    free_poly(sum);
    return 0;
}

/* OUTPUT
Enter Poly 1
Enter the number of terms: 3
Enter the terms, decreasing order of exp
34 2
20 1
5
0
Enter Poly 2
Enter the number of terms: 2
Enter the terms, decreasing order of exp
20 2
5 1
Polynomial A: 34.00x^2 + 20.00x^1 + 5.00x^0
Polynomial B: 20.00x^2 + 5.00x^1
Sum (A + B): 54.00x^2 + 25.00x^1 + 5.00x^0
*/
```

```
FUNCTION. add_poly (p1, p2).
    sum ← creat_poly()
    WHILE (p1->curr != NULL && p2->curr != NULL)
        SWITCH (compare(p1->curr->enp, p2->curr->enp)
            CASE 0 :
                attach(sum, create_term (
                    p1->curr->coef + p2->curr->coef,
                    p1->curr->enp
                )).
                p1->curr ← p1->curr->next.
                p2->curr ← p2->curr->next.
            CASE 1:
                attach(sum, create_term (p1->curr->coef.
                    p1->curr->enp))
                p1->curr ← p1->curr->next
            CASE -1 :
                attach(sum, create_term (p2->curr->coef.
                    p2->curr->enp))
                p2->curr ← p2->curr->next
    ENDWHILE.
    WHILE (p1->curr != NULL).
        attach (sum, create_term (p1->curr->coef.
            p1->curr->coef)).
        p1->curr ← p1->curr->next.
    ENDWHILE.
    WHILE (p2->curr != NULL)
        attach (sum, creat_term (p2->curr->coef.
            p2->curr->enp))
        p2->curr ← p2->curr->next.
```

```
ENDWHILE

FUNCTION free_poly (p)
    tmp ← p->terms.
    WHILE (tmp != NULL)
        ntmp ← tmp.
        tmp ← tmp->next.
        free (ntmp).
    ENDWHILE
    free (p).

OUTPUT ("Enter Poly 1")
p1 ← read_poly().
OUTPUT("Enter Poly 2").
p2 ← read_poly()

Sum ← add_poly(p1, p2)
OUTPUT("Polynomial A")
display_poly(p1)

OUTPUT("Polynomial B")
display_poly(p2)
OUTPUT("Sum: (A+B): ")
display_poly(sum).

free_poly(p1).
free_poly(p2).
free_poly(sum).
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

RETURN

Obtained and verified the output

28/11/22

```c
// binary_search_tree
// VML21AD004
// Binary search tree operations

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


typedef struct nd {
    int data;
    struct nd *left;
    struct nd *right;
} Node;


Node *create_node(int data) {
    Node *new_node = malloc(sizeof(Node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}


void display_tree(Node *root, int level) {

    if (root == NULL) return;

    for (int i = 0; i < level; i++) {
        printf((i == level - 1) ? "|-" : "  ");
    }
    printf("%d\n", root->data);
    display_tree(root->left, level + 1);
    display_tree(root->right, level + 1);
}

Node *smallest_node(Node *root) {
    if (root->left == NULL) return root;
    return smallest_node(root->left);
}


Node *insert_node(Node *root, int data) {
    if (root == NULL) {
        return create_node(data);
    }

    if (root->data == data) {
        printf("Duplicate data not allowed");
        return NULL;
    }

    if (data < root->data)
        root->left = insert_node(root->left, data);
    else if (data > root->data)
        root->right = insert_node(root->right, data);

    return root;
}


Node *delete_node(Node *root, int data) {
    if (root == NULL) return root;

    if (root->data > data) {
        root->left = delete_node(root->left, data);
```

28/11/22

Binary Search Tree

Ex:12

AIM:

Implementation of binary search trees' creations, insertions, deletion and traversal using linked list.

PSEUDOCODE

```
BEGIN.
    Struct Node
        data.
        left
        right.

    FUNCTION create_node (data)
        new_node ← malloc (sizeof (node)).
        new_node -> data ← data.
        new_node -> left ← NULL
        new_node -> right ← NULL
        RETURN new_node.

    FUNCTION display_tree (root, level)
        IF (root == NULL) THEN
            RETURN.
        ENDIF.
        FOR (int i=0 TO level-1).
            OUTPUT ((i == level-1) ? "|-" : " ")
        ENDFOR.
```

```
    } else if (root->data < data) {
        root->right = delete_node(root->right, data);
    } else {
        if (root->left == NULL) {
            Node *temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node *temp = root->left;
            free(root);
            return temp;
        }

        Node *temp = smallest_node(root->right);
        root->data = temp->data;
        root->right = delete_node(root->right, temp->data);

    }
    return root;
}


void free_tree(Node *root) {
    if (root == NULL) return;
    free_tree(root->left);
    free_tree(root->right);
    free(root);
}


Node *create_tree(Node *root) {
    int n_nodes, val;
    printf("Enter the number of nodes: ");

    scanf("%d", &n_nodes);
    printf("Enter the nodes\n");

    for (int i = 0; i < n_nodes; i++) {
        scanf("%d", &val);

        if (root == NULL) {
            root = insert_node(root, val);
            continue;
        }
        insert_node(root, val);
    }
    return root;
}

void preorder_traversal(Node *root) {
    if (root == NULL) return;

    printf("%d ", root->data);
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}

int main() {
    Node *root = NULL;
    int con = 1, c_tree = 0;
    int choice, val;
    while (con) {
        printf("1. Create 2. Insert 3. Delete 4. Traverse\n");
        printf("5. Display 6. Quit\n");

        if (choice == 6) {
```

```
display_tree (root->left, level+1)
display_tree (root->right, level+1)

FUNCTION smallest_node (root).
    IF (root->left == NULL) THEN.
        RETURN smallestnode (root->left).
    ENDIF

FUNCTION insert_node (root, data)
    IF (root == NULL) THEN.
        RETURN (create_node (data)
    ENDIF.
    IF (root->data == data) THEN.
        OUTPUT("Duplicate data not allowed").
        RETURN NULL.
    ENDIF
    IF (data < root->data) THEN.
        root->left <- insert_node (root->left,
                                   data)
    ELSE IF (data > root->data) THEN.
        root->right <- insert_node (root->right,
                                    data).
    ENDIF
    RETURN root.

FUNCTION. delete_node (root, data).
    IF (root == NULL) RETURN root
    ENDIF
```

```
            con = 0;
            continue;
        }

        printf(">>> ");
        scanf("%d", &choice);
        if (choice != 1 && c_tree == 0) {
            printf("Create the tree first\n");
            continue;
        }

        switch (choice) {
            case 1:
                if (c_tree == 0) {
                    root = create_tree(root);
                    c_tree = 1;
                } else {
                    printf("Tree has created already\n");
                }
                break;
            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &val);
                if (root == NULL) {
                    root = insert_node(root, val);
                } else {
                    insert_node(root, val);
                }
                break;
            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &val);
                root = delete_node(root, val);
                break;
            case 4:
                printf("Preorder Traversal of the tree\n");
                preorder_traversal(root);
                printf("\n");
                break;
            case 5:
                display_tree(root, 0);
                break;
            case 6:
                con = 0;
                break;
            default:
                printf("Invalid Input\n");
                break;
        }

    }

    free_tree(root);
    return 0;
}

/* OUTPUT

1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 1
Enter the number of nodes: 8
Enter the nodes
10 6 14 3 8 13 15 12
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
```

```
IF (root -> data > data) THEN.
    root -> left <- delete_node (root -> left,
                                 data).
ELSE IF (root -> data < data) THEN.
    root -> right <- delete_node (root -> right,
                                  data)
ELSE
    IF (root -> left == NULL) THEN.
        free (root)
        RETURN temp.
    ELSE IF (root -> right == NULL) THEN.
        temp <- root -> left.
        free (root)
        RETURN. temp.
    ENDIF.
    temp <- Smallest node (root -> right)
    root -> data <- temp -> data.
    root -> right <- delete_node (root -> right,
                                  temp -> data).

ENDIF.
RETURN root.

FUNCTION free_tree (root).
IF (root == NULL) THEN.
    RETURN
ENDIF
free_tree (root -> left)
free_tree (root -> right).
free (root).
```

```
>>> 5
10
|-6
 |-3
 |-8
|-14
 |-13
  |-12
 |-15
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 2
Enter the value to insert: 11
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 5
10
|-6
 |-3
 |-8
|-14
 |-13
  |-12
   |-11
 |-15
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 3
Enter the value to delete: 6
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 5
10
|-8
 |-3
|-14
 |-13
  |-12
   |-11
 |-15
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 4
Preorder Traversal of the tree
10 8 3 14 13 12 11 15
1. Create 2. Insert 3. Delete 4. Traverse
5. Display 6. Quit
>>> 6

*/
```

```
FUNCTION Create_tree(root)
    OUTPUT("Enter the number of nodes:")
    n_nodes ← USERINPUT
    OUTPUT("Enter the nodes")

    FOR(I=0 TO n_nodes-1)
        val ← USERINPUT
        IF (root == NULL) THEN
            root ← Insert_node(root, val):
            CONTINUE
        ENDIF.
        Insert_node(root, val).
    ENDFOR.
    RETURN root.

FUNCTION preorder_traversal(root)
    IF (root==NULL) THEN.
        RETURN.
    ENDIF.
    OUTPUT(root -> data).
    preorder_traversal(root->left).
    preorder_traversal(root-> right).

root ← NULL.
con ← 1
C_tree ← 0.
WHILE(con)
    OUTPUT("1. Create 2. Insert 3. Delete 4. Traversal")
    OUTPUT(5. Display 6. Quit):
```

```
IF (choice == 6) THEN.
    CON ← 0
    CONTINUE.
ENDIF.
OUTPUT(">>>>").
CHOICE ← USERINPUT

IF (Choice != 1 && C_tree == 0) THEN.
    OUTPUT("Create the tree first")
    CONTINUE.
ENDIF.

SWITCH (choice)
    CASE 1 :
    IF(C_tree == 0) THEN.
        root ← create_tree (root)
        C_tree ← 1
    ELSE
        OUTPUT("Tree has been created already")
    ENDIF.
    CASE 2 :
    OUTPUT("Enter the value to insert").
    val ← USERINPUT.
    IF(root == NULL) THEN
        root = insertnode (root, val);
    ELSE
        insert node (root, val).
    ENDIF.
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

CASE 3:
OUTPUT("Enter the value to delete").
Val ← USERINPUT.
root ← delete_node(root, Val);
CASE 4:
OUTPUT("Pre order traversal of the tree").
preorder_traversal(root).
CASE 5:
display_tree(root, 0)
CASE 6:
Con ← 0
DEFAULT
OUTPUT("Invalid Input").
ENDSWITCH.

free_tree(root).

RESULT:
Obtained and verified the output.

28/11/22

```c
// sort.c
// VML21AD004
// implement different sort algorithms in c

#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// ### selection sort ###

void selection_sort(int arr[], int n) {
    int min_idx;  // the index of the min value;
    for (int i = 0; i < n - 1; i++) {
        min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) min_idx = j;
        }

        if (min_idx == i) continue;
        swap(&arr[i], &arr[min_idx]);
    }
}

// ### insertion sort ###

void insertion_sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            swap(&arr[j], &arr[j - 1]);
            j--;
        }
    }
}

// ### quick sort ###

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[j], &arr[i]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
void quick_sort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}

// ### merge sort ###

void merge(int arr[], int l, int m, int r) {
    int left_l = m - l + 1;
    int right_l = r - m;
```

17/10/22

Sorting.

46

Exp13

AIM:

To implement a C program to sort a given array using Bubble sort, Insertion sort, Selection sort, merge sort, quick sort.

PSEUDOCODE

START.

```
FUNCTION swap (*a, *b)
        temp ← *a
        *a ← *b.
        *b ← temp.

FUNCTION selection sort (arr[], n)
        FOR i = 0 TO n-1
            min_idx ← i
            FOR j = i+1 TO n-1
                IF arr[j] < arr[min_idx] THEN
                    min_idx = j
                ENDIF
            ENDFOR
            swap(&arr[i], &arr[min_idx])
        ENDFOR.
```

```
    // temporary sub arrays
    int temp_l[left_l];
    int temp_r[right_l];
    int i, j, k;

    // copy the value from a to the two sub arrays
    for (int i = 0; i < left_l; i++) {
        temp_l[i] = arr[l + i];
    }
    for (int i = 0; i < right_l; i++) {
        temp_r[i] = arr[m + i + 1];
    }

    // merge the two arrays
    for (i = 0, j = 0, k = l; k <= r; k++) {
        if ((i < left_l) && (j >= right_l || temp_l[i] <= temp_r[j])) {
            arr[k] = temp_l[i++];
        } else {
            arr[k] = temp_r[j++];
        }
    }
}
void merge_sort(int arr[], int l, int r) {
    if (l < r) {
        int mid = (l + r) / 2;
        merge_sort(arr, l, mid);
        merge_sort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }
}

// ### bubble sort ###

void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
            }
        }
    }
}

int main() {
    int n_terms;
    printf("Enter the number of elements: ");
    scanf("%d", &n_terms);

    int arr[n_terms];

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

    printf("Enter the sorting algorithm\n");
    printf("1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection
Sort\n>>> ");

    int choice;
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Bubble sort\n");
```

Signature Not Verified

Principal.VJEC

```
FUNCTION insertion_sort (arr[], n)
    FOR i=1 TO i<n-1
        j ← i
        WHILE (j>0 && arr[j-1] > arr[j])
            swap(&arr[j], &arr[j-1])
            j ← j-1
        ENDWHILE
    ENDFOR.

FUNCTION partition (arr[], low, high)
    pivot ← arr[high]
    i ← low - 1
    FOR j = low TO high-1
        IF (arr[j] < pivot) THEN
            i ← i+1
            swap (&arr[i], &arr[j])
        ENDIF.
    ENDFOR.

FUNCTION quick_sort(arr[], low, high).
    IF ( low < high) THEN.
        pi ← partition (arr, low, high).
        quicksort (arr, low, pi-1);
        quicksort(arr, pi+1, high)
    ENDIF.
```

Signature Not Verified

Principal.VJEC

```
            bubble_sort(arr, n_terms);
            break;
        case 2:
            printf("Insertion sort\n");
            insertion_sort(arr, n_terms);
            break;
        case 3:
            printf("Quick sort\n");
            quick_sort(arr, 0, n_terms - 1);
            break;
        case 4:
            printf("Merge sort\n");
            merge_sort(arr, 0, n_terms - 1);
            break;
        case 5:
            printf("Selection sort\n");
            selection_sort(arr, n_terms);
            break;
        default:
            printf("Invalid Input\n"); .
            break;
    }

    printf("The sorted array\n");

    for (int i = 0; i < n_terms; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}

/* OUTPUT

Enter the number of elements: 5
Enter 5 elements
0 2 1 5 7
Enter the sorting algorithm
1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection Sort
>>> 1
Bubble sort
The sorted array
0 1 2 5 7

Enter the number of elements: 5
Enter 5 elements
0 2 1 5 7
Enter the sorting algorithm
1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection Sort
>>> 2
Insertion sort
The sorted array
0 1 2 5 7

Enter the number of elements: 5
Enter 5 elements
0 2 1 5 7
Enter the sorting algorithm
1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection Sort
>>> 3
Quick sort
The sorted array
0 1 2 5 7

Enter the number of elements: 5
Enter 5 elements
```

```
FUNCTION merge (arr[], l, m, r)
    left_l ← m-l+1
    right_l ← r-m

    temp l[left_l]
    temp r[right_l]

    FOR i=0 TO left_l-1
        temp_l[i] = arr[l+i].
    ENDFOR.
    FOR i=0 TO right_l-1
        temp_r[i] = arr[m+i+1]
    ENDFOR

    FOR( i=0, j=0, k=l TO r).
        IF((i< left_l) && (j>=right_l || temp_l[i]
            <= temp_r[j])) THEN.
            arr[k] = temp_l[i]
            i ← i+1.
        ELSE
            arr[k] = temp_r[j]
            j ← j+1
        ENDIF
    ENDFOR.

FUNCTION merge_sort (arr, l, r)
    IF( l<r) THEN
        mid ← (l+r)/2.
        merge_sort (arr, l, mid).
```

```
0 2 1 5 7
Enter the sorting algorithm
1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection Sort
>>> 4
Merge sort
The sorted array
0 1 2 5 7

Enter the number of elements: 5
Enter 5 elements
0 2 1 7 5
Enter the sorting algorithm
1. Bubble Sort 2. Insertion Sort 3. Quick Sort 4. Merge Sort 5. Selection Sort
>>> 5
Selection sort
The sorted array
0 1 2 5 7

*/
```

```
        merge_sort(arr, mid+1, r)
        merge(arr, l, mid, r)
    ENDIF.

FUNCTION bubble_sort(arr[], n).
    FOR (i=0 TO n-1)
        FOR j=0 TO n-i-1
            IF (arr[j] > arr[j+1]) THEN
                swap(&arr[j], &arr[j+1])
            ENDIF.
        ENDFOR.
    ENDFOR

OUTPUT("Enter the number of terms")
n_terms ← USERINPUT
OUTPUT("Enter", n_terms, "elements")
FOR (i=0 TO n_terms-1).
    arr[i] ← USERINPUT.
ENDFOR.

OUTPUT("Enter the sorting algorithm").
OUTPUT(" 1. Bubble Sort 2. Insertion Sort 3. Quick Sort
    4. merge Sort 5. Selection Sort (n>>7)");

choice ← USERINPUT.
SWITCH (choice)
    CASE 1:
        OUTPUT("Bubble Sort")
        bubble_sort(arr, n_terms).
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	**Algorithm**	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	**Program**	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	**Output**	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	**Time taken**	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	**Record**	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

```
CASE 2:
    OUTPUT ("Insertion sort")
    Insertion_sort(arr, n_terms)
CASE 3:
    OUTPUT ("Quick sort")
    quick_sort (arr, n_terms)
CASE 4:
    OUTPUT ("merge Sort")
    merge_sort (arr, 0, n_terms -1)
CASE 5:
    OUTPUT ("selection sort")
    selection_sort (arr, n_terms):
DEFAULT:
    OUTPUT ("Invalid Input"):
ENDSWITCH.
OUTPUT ("The sorted array")
FOR (i=0 TO n_terms -1)
    OUTPUT (arr[i]).
ENDFOR.
```

RESULT.

Obtained and verified the output.

13/12/22

Signature Not Verified

Principal.VJEC

```c
// bfs.c
// VML21AD004
// Breadth first search on a graph in C

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

#define MAX_SIZE 100

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

typedef struct item Item;

Item *queue = NULL;
int visited[MAX_SIZE];
int am[MAX_SIZE][MAX_SIZE];
int n_visited = 0;

void q_enq(int data) {
    Item *new_item = malloc(sizeof(Item));
    new_item->data = data;
    new_item->next = NULL;

    if (queue == NULL) {
        queue = new_item;
        return;
    }

    Item *temp = queue;
    while (temp->next != NULL) temp = temp->next;

    temp->next = new_item;
}

int q_deq() {
    if (queue == NULL) return -1;

    int data = queue->data;
    Item *temp = queue;
    queue = queue->next;
    free(temp);
    return data;
}

int is_visited(int val) {
    int r = 0;
    for (int i = 0; i < n_visited; i++) {
        if (visited[i] == val) {
            r = 1;
            break;
        }
    }
    return r;
}

void bfs(int n_vertex, int start_node) {
    q_enq(start_node);

    while (queue != NULL) {
        int val = q_deq();
        // add to visited
        visited[n_visited++] = val;
```

Signature Not Verified

Principal.VJEC

19/12/22

BFS Traversal.

Exp 14

To implement Breadth first Search on Graph. in C.

PSEUDOCODE

ALGORITHM.

1 BEGIN.

```
STRUCT item
        data.
        next.

queue ← NULL
n_visited ← 0.

FUNCTION q_enq (data)
        new_item = malloc (sizeof (Item)).
        new_item -> data ← data.
        new_item -> next ← NULL.

        IF (queue == NULL) THEN
                queue ← new_item.
                RETURN.
        ENDIF.
        temp ← queue.
        WHILE (temp -> next != NULL)
                temp ← temp -> next.
        ENDWHILE
```

```
        for (int j = 0; j < n_vertex; j++) {
            if (am[val - 1][j] == 1) {
                if (!is_visited(j + 1)) {
                    q_enq(j + 1);
                }
            }
        }
    }

    if (n_visited != n_vertex) {
        for (int i = 1; n_vertex + 1; i++) {
            if (!is_visited(i)) {
                bfs(n_vertex, i);
                break;
            }
        }
    }
}

void read_edges() {
    int n_edges;
    printf("Enter the number of edges: ");
    scanf("%d", &n_edges);

    int vi, vj;

    printf("Enter %d the edges (start end)\n", n_edges);
    for (int i = 0; i < n_edges; i++) {
        scanf("%d%d", &vi, &vj);

        am[vi - 1][vj - 1] = 1;
    }
}

void print_am(int len) {
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            printf("%d ", am[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main() {
    int n_vertex;
    printf("Enter the number of vertexes: ");
    scanf("%d", &n_vertex);

    // zero out the matrix
    for (int i = 0; i < n_vertex; i++) {
        for (int j = 0; j < n_vertex; j++) {
            am[i][j] = 0;
        }
    }

    // read edges
    int n_edges;
    int start_node;
    read_edges();
    printf("The adjacency matrix\n");
    print_am(n_vertex);

    printf("Enter the node to start the search from: ");
    scanf("%d", &start_node);
```

```
temp->next = new_item.

FUNCTION q_deq().
    IF (queue == NULL) THEN.
        RETURN -1
    ENDIF
    data <- queue->data.
    temp <- queue.
    queue <- queue->next
    free(temp).
    RETURN data.

FUNCTION is_visited(val)
    n <- 0.
    FOR (i = 0 TO n_visited-1)
        IF (visited[i] == val) THEN.
            n = 1
            BREAK
        ENDIF.
    ENDFOR.
    RETURN n.

FUNCTION bfs (n_verten, start_node)
    q_enq(start_node).
    WHILE (queue != NULL)
        val <- q_deq()
        visited[n_visited++] <- val.
        FOR(j = 0 TO n_verten-1)
            IF (!is_visited(j+1)) THEN.
```

```
    bfs(n_vertex, start_node);

    for (int i = 0; i < n_vertex; i++) {
        printf("%d ", visited[i]);
    }
    printf("\n");

    return 0;
}


/* OUTPUT

Enter the number of vertexes: 5
Enter the number of edges: 6
Enter 6 the edges (start end)
1 2
2 3
3 4
4 5
5 1
1 4
The adjacency matrix
0 1 0 1 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
1 0 0 0 0

Enter the node to start the search from: 1
1 2 4 3 5

*/
```

```
                q.enq(j+1).
            ENDIF
        ENDIF.
    ENDFOR.
ENDWHILE

IF (n_visited != n_vertex) THEN
    FOR (i = 1 TO n_vertex+1)
        IF (!isVisited(i)) THEN
            bfs(n_vertex, i)
            break
        ENDIF.
    ENDFOR.
ENDIF.

FUNCTION read_edges()
    OUTPUT("Enter the number of edges:").
    n_edges ← USERINPUT.

    OUTPUT("Enter the edges (start end)").
    FOR(i = 0 TO n_edges-1)
        vi ← USERINPUT
        vj ← USERINPUT.
        adm[vi-1][vj-1] = 1;
    ENDFOR.

FUNCTION print_adm(len)
    FOR(i = 0 TO len-1)
        FOR(j = 0 To len-1).
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.
Score for this experiment (30)					
Viva (15)					
Total Marks (45)					
Signature					

Signature Not Verified

Principal.VJEC

```
                OUTPUT (am[i][j]).
            ENDFOR.
        ENDFOR.

OUTPUT ("Enter the number of vertices:")
n_vertex ← USERINPUT

FOR (i=0 TO n_vertex-1)
    FOR (j=0 TO n_vertex - 1)
        am[i][j]=0
    ENDFOR.
ENDFOR

read_edges()
OUTPUT("The adjacency matrix")
print am (n_vertex)
OUTPUT("Enter the node to start the search from:")
start node ← USERINPUT.
bfs(n_vertex, start_node)

FOR(i=0 To n_vertex-1)
        OUTPUT (visited[i])
ENDFOR.

END.
```

RESULT

Obtained and verified the output.

```c
// dfs.c
// VML21AD004
// Depth first search on a graph in C

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

#define MAX_SIZE 100

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

typedef struct item Item;

Item *stack = NULL;
int visited[MAX_SIZE];
int am[MAX_SIZE][MAX_SIZE];
int n_visited = 0;

void s_push(int data) {
    Item *new_item = malloc(sizeof(Item));
    new_item->data = data;
    new_item->next = NULL;

    if (stack == NULL) {
        stack = new_item;
        return;
    }

    Item *temp = stack;
    while (temp->next != NULL) temp = temp->next;

    temp->next = new_item;
}

int s_pop() {
    if (stack == NULL) return -1;
    Item *temp = stack;

    if (stack->next == NULL) {
        int t = stack->data;
        free(stack);
        stack = NULL;
        return t;
    }

    Item *prev;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    int val = temp->data;
    free(temp);
    prev->next = NULL;
    return val;
}

int is_visited(int val) {
    int r = 0;
    for (int i = 0; i < n_visited; i++) {
        if (visited[i] == val) {
            r = 1;
            break;
        }
```

17/12/22

DFS Traversal

EX NO: 15

AIM

To implement Depth First Search on a graph in C.

PSEUDOCODE

```
BEGIN.
    Struct item.
        data.
        next.

    Stack ← NULL
    nvisited ← 0

    FUNCTION Spush(data).
        new_item = malloc(sizeof(item))
        new_item->data ← data.
        new_item->next ← NULL

        IF (Stack == NULL) THEN.
            Stack ← new_item.
            RETURN.
        ENDIF
        temp ← Stack.
        WHILE (temp->next != NULL)
            temp ← temp->next.
        ENDWHILE
```

```
    }
    return r;
}

void dfs(int n_vertex, int start_node) {
    s_push(start_node);

    while (stack != NULL) {
        int val = s_pop();
        // add to visited
        visited[n_visited++] = val;

        for (int j = 0; j < n_vertex; j++) {
            if (am[val - 1][j] == 1) {
                if (!is_visited(j + 1)) {
                    s_push(j + 1);
                }
            }
        }
    }

    if (n_visited != n_vertex) {
        for (int i = 1; n_vertex + 1; i++) {
            if (!is_visited(i)) {
                dfs(n_vertex, i);
                break;
            }
        }
    }
}

void read_edges() {
    int n_edges;
    printf("Enter the number of edges: ");
    scanf("%d", &n_edges);

    int vi, vj;

    printf("Enter %d the edges (start end)\n", n_edges);
    for (int i = 0; i < n_edges; i++) {
        scanf("%d%d", &vi, &vj);

        am[vi - 1][vj - 1] = 1;
    }
}

void print_am(int len) {
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            printf("%d ", am[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

int main() {
    int n_vertex;
    printf("Enter the number of vertexes: ");
    scanf("%d", &n_vertex);

    // zero out the matrix
    for (int i = 0; i < n_vertex; i++) {
        for (int j = 0; j < n_vertex; j++)
            am[i][j] = 0;
        }
```

```
FUNCTION S_pop().
    IF(stack == NULL) THEN.
        return -1
    ENDIF
    temp ← stack.

    IF(stack->next == NULL) THEN.
        t ← stack->data.
        free(stack).
        stack = NULL
        return t.
    ENDIF
    WHILE (tem->next != NULL)
        prev ← temp
        temp ← tem->next
    ENDWHILE.
    val ← temp->data.
    free (temp).
    prev->next ← NULL.
    return val.

FUNCTION is_visited (val)
    n ← 0
    FOR (i = 0 TO n_visited-1)
        IF(visited[i] == val) THEN
            n ← 1
            break.
        ENDIF
    ENDFOR
```

```
    }

    // read edges
    int n_edges;
    int start_node;
    read_edges();
    printf("The adjacency matrix\n");
    print_am(n_vertex);

    printf("Enter the node to start the search from: ");
    scanf("%d", &start_node);

    dfs(n_vertex, start_node);

    for (int i = 0; i < n_vertex; i++) {
        printf("%d ", visited[i]);
    }
    printf("\n");

    return 0;
}

/* OUTPUT

Enter the number of vertexes: 5
Enter the number of edges: 6
Enter 6 the edges (start end)
1 2
2 3
3 4
4 5
5 1
1 4
The adjacency matrix
0 1 0 1 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
1 0 0 0 0

Enter the node to start the search from: 1
1 4 5 2 3

*/
```

return n.

FUNCTION dfs (n_vertex, startnode)
S_push (startnode)
WHILE (stack != NULL)
val ← S_pop()
visited[n_visited+1] ← val
FOR(j=0 TO j<n_vertex-1)
IF(adjval-1][j] == 1) THEN
IF(!(is_visited(j+1)) THEN
S_push(j+1)
ENDIF
ENDIF
ENDFOR
ENDWHILE

IF(n_visited != n_vertex) THEN
FOR (i=1 TO n_vertex+1)
IF(!is_visited(i)) THEN
dfs(n_vertex, i)
break;
ENDIF
ENDFOR
ENDIF

FUNCTION read_edges()
OUTPUT("Enter the number of edges")
n_edges ← USERINPUT

```
OUTPUT("Enter the edges (start end)")
FOR (i = 0 to n_edges - 1)
        vi ← USERINPUT
        vj ← USERINPUT
        am[vi - 1][vj - 1] = 1
ENDFOR.

FUNCTION print_am (len)
        FOR (i = 0 to len - 1)
            FOR(j = 0 TO len - 1).
                OUTPUT(am[i][j]).
            ENDFOR.
        ENDFOR

OUTPUT("Enter the number of vertices").
n_vertex ← USERINPUT.

FOR (i = 0 To n_vertex - i)
        FOR(j = 0 TO n_vertex - 1)
            am[i][j] = 0.
        ENDFOR
ENDFOR.
read_edges()
OUTPUT("The adjacency matrix"):
print_am(n_vertex)
OUTPUT("Enter the node to start the search from")
start_node ← USERINPUT
dfs(n_vertex, start_node).
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Tot
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

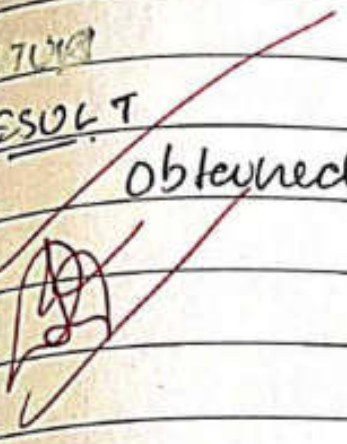
Principal.VJEC

```
FOR (i = 0 TO n-vertex - 1)
        OUTPUT (visited[i])
ENDFOR.
```

RETURN

RESULT

Obtained and verified output.



Completed & Certified

Rajitha. K. V.

20/12/22.

Signature Not Verified

Principal.VJEC

LABORATORY RECORD

Name ..... Sheethal cP .....

Semester & Branch ..... S4 CSE -C .....

Roll No. ..... 51 .....

University Reg. No. ..... VML20CS153 .....

Certified that this is the bonafide record of the work done in

the ..... Operating System ..... *laboratory of*

Vimal Jyothi Engineering College, Chemperi, Kannur

by

Mr. / Ms. ..... SHEETHAL C P .....

Date ..... 29/07/2022 .....

Place ..... Chemperi .....

Staff in charge

HEAD OF THE DEPARTMENT
Dept. of Computer science & Engg.
Vimal Jyothi Engineering College
Chemperi-670 632

Head of the Department

10/10/22

Internal Examiner

10/10/22

External Examiner

Signature Not Verified

Principal.VJEC

INDEX

Exp. No.	Date	Name of the experiment	Page No.	Marks	Signature of the faculty
1	9/05/2022	Basic Linux Commands	1	45	
2.	19/05/2022	Shell Programming	4	45	
3	06/06/2022	System calls	19	43	
4	23/06/2022	I/O System call	30	45	
5	30/06/2022	Inter Process communication using shared memory	36	42	
6	30/06/2022	Semaphore	41	43	29/07/22
7	4/07/2022	CPU scheduling	45	45	
8	21/7/2022	Memory allocation Methods	55	45	
9	21/07/2022	Page Replacement Algorithm	60	44	
10	25/07/2022	Banker's Algorithm	64	45	
11	25/07/22	Disk Scheduling Algorithm	67	45	

RUBRICS FOR EVALUATION OF FAMILIARIZATION EXPERIMENTS

Sl. No.	Performance criteria	Excellent - 10	Good - 7	Satisfactory -4	Poor -1	Total
1	Basic Linux commands and system calls identification and execution	Identified and executed all the commands and its functions correctly.	Executed all the commands, but its functions are not clear.	Executed more than 50% or more of the commands and its functions.	Executed less than 50% of the commands and its functions.	10
2	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	10
3	Record	Neat, well labeled, organized and completed within time	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	10
Total score (30)						30
Viva (15)						15
Total (45)						45
Signature						Nayana

Experiment No : 1

Date : 9/05/2022

Linux Commands

Aim

Write basic linux command

Program

File commands		
1.	ls	Directory listing
2.	ls -al	Formatted listing with hidden files
3.	ls -lt	Sorting the formatted listing by time modification
4.	cd dir	change directory to dir
5.	cd	change to home directory
6.	Pwd	Show current working directory
7.	mkdir dir	Creating a directory dir
8.	cat > file	Places the standard input into the file
9.	more file	Output the content of the file
10.	head file	output the first 10 lines of the file
11.	tail file	output the last 10 lines of the file
12	tail -f file	output the content of the file as it grows, starting with the last 10 lines
13	touch file	Create or update file
14	rm file	Deleting the file

File commands		
15	rm -r dir	Deleting the directory
16	rm -f file	Force to remove the file
17	rm -rf dir	Force to remove directory dir
18	cp file1 file2	Copy the contents of the file1 to file 2
19	cp -r dir1 dir2	copy dir1 to dir2, Create dir2 if not Present
20	mv file1 file2	Rename or move file1 to file2, if file2 is an Existing directory

Result

Output is Successfully obtained and verified

09/02/22

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
e for this experiment (30)					30
(15)					15
l Marks (45)					45
ature					

Signature Not Verified

Principal.VJEC

Shell Programming

Aim

Write a Program to implement Shell Programming
- Command Syntax
- Write simple functions with basic tests, loops, Patterns

Introduction to Shell Programming

Shell

- Shell is a UNIX term for an interface between a user and an operating System Service
- Shell Provides users with an interface and accepts human-readable commands into the System and Executes those commands which can run automatically and gives the output of Program in a Set shell script
- An operating is made of many components, but its two Prime components are :
  - ⇒ Kernal
  - ⇒ Shell

A Kernal is at the nucleus of a compiler. It makes the communication between the hardware and Software Possible

A kernal is the innermost part of an operating system
A shell is the outermost one

Types of shell

There are two main shell in Linux:
1. Bourne shell
2. C shell

Steps to write shell script in Ubuntu

1. Create a file using gedit editor or any other editor
2. Name script file with Extension .sh
3. Start the script with #!/bin/bash
4. write some code
5. Save the script file as filename.sh
6. For Executing the script type ./filename.sh

Program NO :1

Write a shell program to print "Hello world"

Program

#!/bin/bash

```
echo "Hello World"
```

Output

```
chmod +X hello.sh
./hello.sh

Hello World
```

Program No: 2

write a Shell Program to declare and initialize Varibles

Program

```
#!/bin/bash
NAME="Sheethal"
echo "My name is $NAME"
```

Output:

```
chmod +X Pgm2.sh
./Pgm2.sh

My name is Sheethal
```


Program No: 3

Write a Shell program to read a varible using keyboard

Program

```
#!/bin/bash
read -P "Enter your name" $S '\n' S
Echo "hello, $S"
```

Output

```
chmod +X Pgm2.sh
Enter your name sheethal
  Sheethal
  hello, Sheethal
```

Program 4

Write a Shell Program to read Sum of two numbers

Program

```
#!/bin/bash
read -P "Enter first number" A
read -P "Enter Second number" B
Sum=`expr $A + $B`
Echo "Sum = $Sum"
```

Output

```
chmod -x Pgm4.sh
Enter first number A 4
Enter second number B 6
  Sum = 10
```

Program No: 5

Write a Program to check odd or Even

Program

```
#!/bin/bash
read -p "Enter a number" M
if [ `expr $M % 2` == 0 ]
then
  echo "$M is even"
else
  echo "$M is odd"
fi
```

Output

```
chmod -x Pgm5.sh
Enter a number 3
3 is odd
```

./Pgm5.sh
Enter a number 2

2 is odd

to 5

Program No: 6

Write a Shell Program to find largest among two numbers

Program

```
#!/bin/bash
read -p "Enter the first number" A
read -p "Enter the Second number" B
if [ $A == $B ]
then
  echo "$A and $B are Equal"
elif [ $A -gt $B ]
then
  echo "$A is largest"
else
  echo "$B is largest"
fi
```

Output

```
chmod +x Pgm6.sh
./Pgm6.sh
Enter the first number 4
Enter the Second number 5

5 is Largest

./Pgm6.sh
Enter the first number 4
Enter the Second number 4

4 and 4 are Equal
```

Program NO: 7

Write a Program to check largest among three numbers

Program

```
#!/bin/bash
read -P "Enter the first number" A
read -P "Enter the Second number" B
read -P "Enter the third number" C
if [ $A -gt $B ]
then
```

```
if [ $A -gt $c ]
then
  echo "$A is largest"
else
  echo "$c is largest"
fi
elif [ $B -gt $c ]
then
  echo "$B is largest"
else
  echo "$c is largest"
fi
```

Output

```
chmod +x Pgm7.sh
./Pgm7.sh
Enter the first number 4
Enter the Second number 10
Enter the third number 7

10 is largest
```

Program No: 8

Write a Shell Program to print "welcome" 5 times on

Screen?

Program

```
#!/bin/bash
c=1
while [ $c -le 5 ]
do
  echo "welcome"
c=`expr $c+1`
done
```

output

```
chmod +x pgm8.sh
./pgm8.sh
Welcome
Welcome
Welcome
Welcome
Welcome
```

Pg Program No: 9

Write a shell Prog[illegible] Print first 10 ~~loop~~ number by for loop

Program

```
#!/bin/bash
for ((c=1; c<=10; c++))
do
  echo "$c"
done
```

output

```
chmod +x Pgm9.sh
./Pgm9.sh
1
2
3
4
5
6
7
8
9
10
```

o/p verified

Nayana 19/05/22

Program No : 10

write shell Program to find the factorial of a number

Program

```
#!/bin/bash
read -p "Enter the number" A
F=1
for ((i=1; i<=A; i++))
do
F=$(($F*$i))
done
echo "Factorial of $A is $F"
```

Output-

```
chmod +x pgm10.sh
./pgm10.sh

Enter the number
5
Factorial of 5 is 120
```

Program 11

Write a Shell Program to Print Febonacci Series

Program

```
#!/bin/bash
read -p "Enter the number of terms" n
t1=0
t2=1
echo "Febonacci Series"
echo -n "$t1 $t2 "
for ((i=3; i<=n; i++))
do
t3=`expr $t1 + $t2`
echo -n "$t3"
t1=$t2
t2=$t3
done
echo $'\n'
```

Output

```
chmod +x pgm11.sh
./pgm11.sh
```

Enter the number of terms
5
Fibonacci series
0 1 1 2 3

Program No : 12

Write a shell program to print the following pattern

```
*
* *
* * *
* * * *
. . . . . .
- - - - - - - - -
```

Program

```
#!/bin/bash
read -p "Enter the number of rows" n
for (( i=1; i<=n; i++ ))
do
for (( j=1; j<=i; j++ ))
do
Echo -n "* "
done
```

```
echo $'\n'
done
```

Output

```
chmod +x pgm12.sh
./pgm12.sh
```

```
Enter the number of rows
4
*
* *
* * *
* * * *
```

o/p verified

Result

Familiarized with shell commands, Syntax of for loop, if and while loop

All Shell Programs are Executed Successfully and output obtained

Navya 30/03/22

Algorithm

1. Start
2. Write Shebang Statement
3. Print hello World
4. Stop

Algorithm

Program -2

1. start
2. write Shebang statement
3. Create a variable name assign value 'Sheethal'
4. Print the variable name
5. stop

Program 3

1. Start
2. Write Shebang statement
3. Read a value from keyboard and store it in 's' variable
4. Print the content of 's' variable
5. stop

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						13
Total Marks (45)						43
Signature						

Signature Not Verified

Principal.VJEC

Experiment No: 3
Date : 06/06/2022

System Call

Aim

To understand the System call fork(), getpid(), Exec(), wait(), opendir(), readdir(), close(), stat() and Exist()

1. Fork()

* Fork System call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (Parent Process)
* After a new child process is created, both process will Execute the next instruction following the fork() System call
* A child process uses the same PC (Program counter), same cpu registers, same open files which uses in the parent Process
* It takes no Parameters and return an Integer Value
* # Negative value : creation of child process was unsuccessfull

  Zero : Returned to the newly created child Process

  Positive : Process ID is created

* It is built-in function that are defined in <unistd.h> library.

Total Number of Processes = 2^n, where n is the no. of fork System call

Algorithm

1. start
Step 2: a = fork()
Step 3: if a Equal to 0 then goto Step 4 otherwise goto Step 5
Step 4: Print "Hello from child Process"
Step 5: if a>0, then goto Step 6 else goto Step 7
Step 6: Print "Hello from Parent Process"
Step 7: Print "Process Creation is failed"
Step 8: Stop

2. Getpid()

* getpid() is the function used to get the Process id of the process that calls that function
* The PID for the initial Process is 1, and then each new Process is assigned a new id
* It is a Simple function to getting the PID
* This function only helps you in getting the unique

Process ids

* Two types of IDs are present here. one is the current id of the process PID (Getpid()), whereas the other is the ID of the parent process PPID (Getppid())
* This function helps in returning the id of the process that is currently called
* Both these function are built-in function that are defined in <unistd.h> library

Algorithm

1. start
2. a = fork();
3. if a Equal to 0 then goto step 4 otherwise goto step 5
4. Print ("child Process");
   Print ("Message : Hello");
   Print ("child process id : %d", getpid());
   Print ("child process's parent id : %d", getppid());
5. if a>0, then goto step 6 else goto step 7
6. Print ("Parent Process");
   Print("Message : Hi");
   Print("Parent Process id : %d", getpid());
   Print("Parent Process's Parent id : %d", getppid());
7. Print ("Process creation failed");

8. stop

3. Exec()

* The Exec() family of function replaces the current Process image with a new Process image
* It loads the Program into the current Process Space and runs it form from the Entry Point
* The Exec family has many functions in C
* These C functions are basically used to run a System command in a Seguate separate process that the main Program and Print the output
* The Exec function families are defined in the header <unistd.h>
* The available Exec function along with their function Parametus are given beloco
  - int Execl(const char *path, const char * arg, ..., NULL);
  - int Execlp(const char * file, const char * args, ..., NULL);
  - int Execv(const char *path, char *const argv[]);
  - int Execvp(const char *file, char * const argv[]);
  - int Execle(const char *path, const char *arg, ..., NULL, char * const Envp[]);
  - int Execve(const char * file, char * const argv[], char *const Envp[]);

int Execl(const char *path, const char *arg, ..., NULL);

- In Execl() System function takes the path of the Executable binary file (ie, /bin/ls) as the first and Second argument
- Then, the argument (ie, -lh, /home) that you want to Pass to the Executable followed by NULL

Algorithm

1. Start
2. Initialize Pointer variable path = "/bin/ls"
3. initialize Pointer variable *arg1 = -a
4. initialize pointer variable * arg2 = /home/cglab/Desktop
5. call the function Execl with Path, arg1, arg2, NULL
6. STOP

Wait()

All call to wait() blocks the calleng Process until one of its child Process Exist or a Signal is received
After child Process termenate Parent continues its Execution after wait System call instruction
- child Process may terminate due to any of these
  * it calls Exist()

* It returns (an int) from main
* It receives a Signal (from the OS or another Process) whose default action is to terminate

Algorithm

1. Start
2. a = fork();
3. If a Equal to 0 the goto Step 4
4. Print ("Child Process");
5. If a > 0, then goto Step 6 else goto step 7
6. Print Parent Process then Call function wait and Set Null
7. Print bye
8. stop

opendir()

The opendir() function opens a directory Stream corresponding to the directory name, and returns a Pointer to the directory Stream

- The Stream is positioned at the first Entry in the directory

- It is defined in <dirent.h> header file
- The function take a single char pointer argument to specify the directory name to open

* opendir returns DIR* structure or NULL if an error is encountered

- DIR is a data type implimented to represent directory stream

readdir()

- once the directory stream is open and we retrieved the valid DIR*, we can read each entry in it using the readdir function
- Each call to readdir function return a pointer to dirent structure representing the next directory entry
- When the end of the directory stream is reached, readdir return NULL.T

Algorithm

1. Start
2. Creat a pointer corresponding to dirent structure
3. call opendir function with current working directory has argument

4. Get the return value of opendir function to a pointer variable of type DIR
5. If d = NULL then Print "could not open the current directory" otherwise goto step 6
6. read all the entries in the current directory by using readdir function
   6.1 check whether the return value of readdir function for each entry in the current directory is null or not
   6.2 If it is null then close the directory otherwise goto step 6.3
   6.3 Print the name of each entries in the current directory by accessing the structure variable d_name of dirent structure
   6.4 Repeat these process until the return value is null
7. stop

Stat ()

- Stat () is a Unix system call that returns file attribute about an inode
- The semantics of stat() vary between operating system
- As an example, Unix command is uses this

System call to retrieve information on files that includes

1) atime : time of last access (ls - lu)
2) ~~mst~~ mtime : time of last modification (ls - l)
3) ctime : time of last status change (ls - lc)

- The <sys/stat.h> header defines the structure of the data returned by the function

Algorithm

1. Start
2. Create an object corresponding to stat structure
3. Call stat function with parameters filename and pointer variable which points to the first entry in the file
4. Print file size by accessing the structure variable st size of stat structure
5. Stop

Exist()

The exist() function is used to terminate a process or function calling immediately in the program

* It means any open file of function belonging to

the Process is closed immediately as the Exist() function occured in the Program
- It is defined in the stdlib.h header file
- The Exist(0) function determines the program terminates without any error messages and then the Exist(1) function determines the Program forcefully terminates the Execution Process

Algorithm

1. Start
2. a = fork()
3. If a equal to 0 then goto step 4 otherwise step 5
4. call the function Exist() and print "child Process"
5. Print "Parent process"
6. Print "Bye"
7. stop

Result

All the programs were Executed and verified and familarised System calls fork(), getpid(), exec(), wait(), opendir(), readdir(), close(), stat() and exit().

Name: Sheethal cp
Roll no: 51

Experiment No: 2
Date:06/06/2022

SYSTEM CALLS

Program using fork()

```
#include<stdio.h>
#include<unistd.h>
void main()
{
        int a=fork();
        if(a==0)
        {
          printf("hello from child process");
        }
        else if(a>0)
        {
         printf("hello from parent process");
        }
        else
        {
         printf("proces creation is failed");
       }
}
```

Output:

```
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ gcc pgm2.c -o pgm2.out
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ ./pgm2.out
Hello from parent process
Hello from child process
```

Program using getpid()

```
#include<stdio.h>
#include<unistd.h>
void main()
{
        int a;
        a=fork();
        if(a==0)
        {
                printf("child process");
                printf("Message:Hello");
                printf("child process id:%d",getpid());
                printf("child process's parent id:%d",getppid());

        else if(a>0)
        {
                printf("parent process");
                printf("Message:Hi");
                printf("parent process id:%d",getpid());
                printf("parent process's parent id:%d",getppid());
       }
       else
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	**Algorithm**	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	**Program**	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	**Output**	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	**Time taken**	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	**Record**	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

Experiment No : 4

Date : 23/06/2022

I/O System Calls

Aim

write a program using I/O System calls of Linux operating System (open, read, write)

Open()

Synopsis:

```
# include <sys/types.h>
# include <sys/stat.h>
# include <fcntl.h>
int open(const char *Pathname, int flags);
```

Description:

Given a pathname for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent System calls (read(2), write(2), lseek(2), fcntl(2), etc.) The file descriptor is not currently open for process

* The argument flag must include one of the following access modes: O_RDONLY, O_WRONLY, or O_RDWR. these request opening the file read-only, write-only or read/write, respectively

Read()

- read from a file descriptor

Synopsis

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Description

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

on files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the no. of bytes read. If the current file offset is at or past the end of file, no bytes are read and read() returns zero

Write()

- Write to a file descriptor

Synopsis:

```
#include <unistd.h>
ssize_t write(int fd, const void *buf,
              size_t count);
```

Description

write() writes up to count bytes from the

buffer pointed buf to the file referred to by the file descriptor fd

the ~~size of~~ sizeof function can be used to find the byte count

lseek()

- reposition read/write file offset

Synopsis:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

Description

The lseek() function repositions the offset of the open file associated with the file descriptor fd to the argument offset according to the directive whence as follows:

SEEK_SET :

The offset is set to offset bytes

SEEK_CUR

The offset is set to its current location plus offset bytes

SEEK_END

The offset is set to the size of the file

Plus off_set bytes

Algorithm

1. Start

Close ()

- close a file descriptor

Synopsis:

```
# include <unistd.h>
Int close (int fd);
```

Algorithm

1. Start
2. Initialize a varible fd
3. Initiate a char m with content "Welcome to the world of Programming" with size 50
4. Initiate a character b with size 50
5. Store the values return by function open() with Parameter "Sample.txt" file name and access mode O_RDWR to fd
6. Print file descriptor
7. Checks if fd not Equal to -1 then goto 7.1 else step 8

7.1 Print file opened

7.2 call the function write with fd, m, sizeof(m)

7.3 call the function lseek with parameter fd, 0, and SEEK_SET

7.4 call the function read with parameter fd, b and sizeof(m)

7.5 Print "was written in to my file"

7.6 call the function close(fd) to close the descriptor

Step 8: Print "could not open the file"

step 9: stop

result:

Program Executed Successfully and Verified the output

28/06/22

Name:Sheethal cp
Roll.no:51

Experiment no:4
Date:23/06/2022

I/O SYSTEM CALLS

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include<unistd.h>
void main()
{
        int fd;
        char m[50]="welcome to the world of programming";
        char b[50];
        fd=open("sample.txt",O_RDWR);
        printf("file descriptor=%d",fd);
        if(fd!=-1)
        {
                printf("file opened");
                write(fd,m,sizeof(m));
                lseek(fd,0,SEEK_SET);
                read(fd,b,sizeof(m));
                printf("%s was written in to my file",b);
                close(fd);
        }
        else
        {
                printf("could not open the file");
        }
}
```

verified

Output

```
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ gcc iosys.c -o iosys.out
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ ./iosys.out
file descriptor=3
file opened
welcome to the world of programming was written in to my file
```

Signature Not Verified

Principal.VJEC

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						12
Total Marks (45)						42
Signature						

Signature Not Verified

Principal.VJEC

Experiment NO: 5

Date : 30/06/2022

Inter Process Communication Using Shared Memory

Aim

Write a Program to implement inter Process communication using Shared memory

Shm get ()

```
# include <sys/IPC.h>
# include <sys/shm.h>
Int shmget (key_t key, size_t size, int shmflg);
```

The first parameter specifies the unique number (called key) identifying the shared segment. The second parameter is the size of the shared segment eg: 1024 byte or 2048 bytes. The third parameter specifies the permissions on the shared segment. on success the shmget () function return a valid identifier while on failure it return -1

Shmat ()

```
# include <sys/shm.h>
# include <sys/types.h>
```

void *shmat(int shmid, const void *shmaddr, int shmflg);

shmat() is used to attach the created shared segment with the address space of the calling process. The first parameter here is the identifier which shmget() function return on success. The second parameter is the address where to attach it to the calling process. A NULL value of second parameter means that the system will automatically choose a suitable address. The third parameter is '0' if the second parameter is NULL, otherwise the value is specified by SHM_RND

Algorithm

Program : Receiver Process

1. Start
2. Initialize the variable id
3. Initialize the pointer variable sm
4. Initialize a character with size 50
5. Store the values returned by function shmget to id
6. Print shared memory id

7. Store the value returned by the function shmat to sm
8. Print Process attached at
9. Print the data read from Shared memory
10. Stop

Algorithm

Program : Sender Process

1. Start
2. Initialize the integer variable id
3. Initialize the pointer variable sm
4. Initialize character with size 50
5. Call the function shmget with parameter with 2345, 1024, 0666, IPC_CREAT and store to id
6. Print Shared memory id
7. Call the function shmat with parameter id, NULL, 0 and store to sm
8. Print Process attached
9. Print enter the data to be written to the shared memory
10. Call the function read with parameter 0, d, 50
11. Call the function strcpy with parameter sm, and

d

12. Print the data written

13. Stop

Result

Programs Executed Sucessfully and verified the output

30/06/2021

Experiment No:5
Date: 30/06/2022

INTER PROCESS COMMUNICATION USING SHARED MEMORY

program: Sender Process

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
        int id;
        void *sm;
        char d[50];
        id=shmget(2345,1024,0666| IPC_CREAT);
        printf("shared memory id=%d\n",id);
        sm=shmat(id,NULL,0);
        printf("process attached  at %p\n",sm);
        printf("enter the data to be written to the shared memory\n");
        read(0,d,50);
        strcpy(sm,d);
        printf("the data written %s\n",(char *)sm);
        return 0;
}
```

Output:Sender Process

```
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ gcc sender.c -o sender.out
pglab@pglab-H310M-H:~/Desktop/sheethal/oslab$ ./sender.out
shared memory id=2097165
process attached  at 0xb7f68000
enter the data to be written to the shared memory
Hi_good morning
the data written  Hi_good morning
```


Signature Not Verified

Principal.VJEC

Program: Reciever Process

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

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						13
Total Marks (45)						43
Signature						Nayana

Signature Not Verified

Principal.VJEC

Experiment NO: 6

Date : 30/06/2022

Semaphore

Aim

write a Program to implement Semaphore
write a Program to implement bounded buffer Problem using Semaphore

Algorithm

1. Start
2. initialize mutex =1, full =0, empty =3 and x =0
3. Call the function Producer
4. Call the function consumer(), wait() and signal()
5. initialize variable n
6: Repeat the following steps until Program return an Exist call
   6.1 Print enter the choice 1. Producer 2. consumer 3. Exist
   6.2 read the choice and store it in variable n
   6.3 checks the following cases using switch keyword
      6.3.1 case :1 mutex == 1 and empty != 0
      6.3.2 Call the function Producer
      6.3.3 otherwise Print buffer is full

6.3.1.3 break

6.3.2 case 2: mutex = 1 and full ≠ 0

6.3.2.1 call the function consumer

6.3.2.2 otherwise print buffer is empty

6.3.2.3 break

6.3.3 case 3: Exist (0)

6.3.3.1 breaks

7. Stop

int wait ()

1. start
2. initialize a variable s
3. Set s = s-1
4. return s
5. stop

int signal ()

1. start
2. initialize a variable s
3. Increment Set s = s+1
4. return s
5. stop

Void Producer ()

1. start
2. call the function wait() passing parameter empty and store in empty

3. call the function wait() with parameter mutex and store in mutex
4. set x = x+1
5. Print Producer produces an item
6. call the function signal() with parameter mutex and store in mutex
7. call the function signal() with parameter full and store in full
8. stop

Void consumer()
1. start
2. call the function wait() with parameter full and store in full
3. call the function wait() with parameter mutex and store in mutex
4. Print consumer consumes an item
5. Set x = x-1
6. Call the function signal() with parameter mutex and store in mutex
7. call the function signal() with parameter empty and store in empty
8. stop

Result
30/06/2022
Program executed successfully and verified the output

Experiment No: 6
Roll No: 51

SEMAPHORE

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
void producer();
void consumer();
int wait(int);
int signal(int);
int main()
{
        int n;
      while(1)
        {
        printf("enter your choice 1.producer 2.consumer 3.exit");
        scanf("%d",&n);
        switch(n)
        {
                case 1:if((mutex==1)&&(empty!=0))
                        {
                                producer();
                        }
                        else
                        {
                                printf("buffer is full");
                        }
                        break;
                case 2:if((mutex==1)&&(full!=0))
                        {
                                consumer();
                        }
                        else
                        {
                                printf("buffer is empty");
                        }
                        break;

                case 3: exit(0);
                        break;
        }
        }
        return 0;
}
int wait(int s)
{
        s--;
        return s;
}
int signal(int s)
{
        s++;
        return s;
}
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Experiment No: 7

Date : 4/07/2022

CPU Scheduling

Aim

Write a Program to implement CPU scheduling algorithm

a) Round Robin

b) SJF

c) FCFS

d) Priority

a) FCFS

1. Start
2. Initialize the variable n, i, at[50], bt[50], p[50], ct[50], tat[50], wt[50], j, k, sum
3. Print Enter the number of Process
4. Set i=0 repeat till i lessthan n
   - 4.1 Set p[i] = i+1
   - 4.2 Print arrival time of Process
   - 4.2 Print Burst time of Process
5. Print process id, arrival time and Burst time
6. Set j=1 repeat till j<n
   - 6.1 Set k=0 repeat till k<n-j
   - 6.2 Sort the table based on the arrival time

6.3 call the Swap function with parameter address P[k] and P[k+1], at[k] and at[k+1], bt[k] and bt[k+1]

7. Set i=0 repeat till i<n

7.1 Set sum = sum + bt[i]

7.2 Set ct[i] = ct[i] + sum

7.3 Set tat[i] = ct[i] - at[i]

7.4 ttat = ttat + tat[i]

7.5 wt[i] = tat[i] - bt[i];

7.6 twt = twt + wt[i]

8. Print after the completion of all process

9. Print Process id, arrival time, Burst time, completion time, turn around time and waiting Time

10. Set i=0 repeat till i<n

11. Print P[i], at[i], bt[i], ct[i], tat[i], wt[i]

12. Set atat = ttat/n

Set awt = twt/n

13. Print Average Turn around time

14. Print Average waiting time

15. Stop

SJF

1. Start

2. Initialize n, i, at[50], bt[50], p[50], ct[50], tat[50], wt[50], j, k, sum

3. declare tat = 0, tcat = 0, twt = 0
4. Print Enter the no. of process
5. Set i = 0 repeat till i < n
   5.1 Set p[i] = i + 1
   5.2 Print arrival time of process
   5.3 Print Burst time of process
6. Print process id, arrival time and Burst time
7. Set i = 0 repeat till i < n
   7.1 Print P[i], at[i], bt[i]
8. Set j = 1 repeat till j < n
   8.1 Set k = 0 repeat till k < n - j
   8.2 check if at[k] > at[k+1]
   8.3 call the Swap function with parameter address p[k] and p[k+1], at[k] and at[k+1], bt[k] and bt[k+1]
9. Set j = 1 repeat till j < n - 1
   9.1 Set k = 0 repeat till k < n - j
   9.2 check if at[k] > at[k+1]
   9.3 call the Swap function with parameter p[k] and p[k+1], at[k] and at[k+1], bt[k] and bt[k+1]
10. Set i = 0 repeat till i < n
   10.1 Set sum = sum + bt[i]
   10.2 Set ct[i] = ct[i] + sum
   10.3 set tat[i] = ct[i] - at[i]
   10.4 ttat = ttat + tat[i]

10.5 wt[i] = tat[i] - bt[i]

10.6 twt = twt + wt[i]

11. Print after the completion of all Process

12. Print Process id, arrival time, completion time, Turnaround time and waiting time

13. Set i=0 repeat till i<n

13.1 Print P[i], at[i], bt[i], ct[i], tat[i], wt[i]

14. Set atat = ttat/n

15 Set awt = twt/n

16 Print average turn around time

17 Print average waiting time

18. Stop

Priority

1. Start

2. Initialize Variables n, i, at[50], bt[50], P[50], ct[50], tat[50], wt[50], j, k, Sum, ps[50]

3. Print Enter the no. of Process

4. Set i=0 repeat till i<n

4.1 Set P[i] = i+1

4.2 Print Arrival time of the Process

4.3 Print Burst time of the process

4.4 Print priority of the Process

5. Print arrival time, burst time and Priority time

6. Set i=0, repeat till i<n
   6.1 Print P[i], at[i], bt[i], px[i]
7. Set j = 1 repeat till j<n
   7.1 Set k=0 repeat till k<n-j
   7.1.1 check whether at[k] > at[k+1]
   7.1.2 call the Swap function with parameter P[k] and P[k+1], at[k] and at[k+1], bt[k] and bt[k+1] and px[k] and px[k+1]
8. Set j=1 repeat till j<n-1
   8.1 Set k=1 repeat till k<n-j
   8.1.1 check whether px[k] > px[k+1]
   8.1.2 call the Swap function with parameter P[k] and P[k+1], at[k] and at[k+1], bt[k] and bt[k+1], px[k], and px[k+1]
9. Set i=0 repeat till i<n
   9.1 Set sum = sum + bt[i]
   9.2 ct[i] = ct[i] + sum
   9.3 tat[i] = ct[i] - at[i]
   9.4 ttat = ttat + tat[i]
   9.5 wt[i] = tat[i] - bt[i].
   9.6 twt = twt + wt[i]
10 Print After the completion of all Processes
11 print Process id, Arrival time, Burst time, completion time, Turn around time and waiting time
12 Set i=0 repeat till i<n

Print P[i], qt[i], bt[i], ct[i], tat[i], wt[i]
14. Set atat = ttat/n
15. Set awt = twt/n
16. Print Average Turnaround time = %0.2f
17. Print Average Waiting time = %0.2f
18. stop

RoundRobin

1. Start
2. Declare queue [100], front = -1
3 - stop

Insert()

1. Start
2. Create a function insert with parameter int m
3. check if front == -1, if true
   3.1 Set front = 0
4. Set rear = rear + 1 and queue[rear] = m
5. Stop

delete()

1. Start

2- Declare variable y as int
3- Set y = queue [Front]
4- Set front = front + 1
5. Stop

Algorithm for main

1. Start
2. Declar n, i, at [50], bt [50], tq, P[50], ct [50] = {0}, temp [50], wt [50], Exist [50] = {0}, l, times = 0 as int, and tat = 0, twt = 0, atat, awt as float
3. Read the no. of process as n
4. Set i = 0, repeat till i < n
   4.1 Set P[i] = i + 1
   4.2 Read the arrival time of process P[i] as at [i]
   4.3 Read the ~~tat~~ burst time of process P[i] as bt [i]
   4.4 Set temp[i] = bt[i]
   4.5 Set i = i + 1
5. Print Process id, Arrival time and Burst time
6. Set i = 0, repeat while i < n
   6.1 Print P[i], at [i] and bt [i]
7. Read the time quantum as tq
8. call insert (0)

9. Set Exist [0] = 1

10. When while (front <= rear), do

10.1 Set i = delete ()

10.2 Checks if (bt[i] >= tq), if true then

10.2.1 Set bt[i] = bt[i] - tq

10.2.2 Set temes = temes + tq

10.3 else

10.3.1 Set times = temes + bt[i]

10.3.2 Set bt[i] = 0

10.4 Set p = 0, repeat the process till n2p

10.4.1 checks if (Exist[i] == 0 and at[i] <= time)

if true then

a) call insert (i)

b) Set Exist[i] = 1

10.5 checks if bt[i] = 0, if true then

10.5.1 Set ct[i] = temes

10.5.2 Set tat[i] = ct[i] - at[i]

10.5.3 Set wt[i] = tat[i] - temp[i]

10.5.4 Set ttat = ttat + tat[i]

10.5.5 Set twt = twt + wt[i]

10.6 Else

10.6.1 call insert (i)

11 Print After the completion of all process

12 Print process id, Arrival time, Burst time,

completion time, Turnaround time and waiting time

13. Set i=0, Repeat the steps till i<n
    13.1 Print P[i], at[i], temp[i], ct[i], tat[i] wt[i]
    13.2 Set i=i+1
14 Set atat = ttat/n
15 Set awt = twt/n
16 Print the average Turn around time, atat
17 Print the average waiting time, awt
18 Stop

Algorithm for Swap

1. Start
2. Creat a function Swap with Parameter pointers Variable x, y
3. Declare temp as *x
4. Set *x as *y
5. Set *y as temp
6. Stop

Result

The Program has been Successfully Executed and output obtained

Experiment n0:7
Date :04/07/2022

sheethal cp
roll no :51

CPU SCHEDULING

FCFS

```
#include<stdio.h>
void swap(int *a,int *b)
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}
void main()
{
        int n,i,at[50],bt[50],p[50],ct[50]={0},tat[50],wt[50],j,k,sum=0;
        float ttat=0,tcat=0,twt=0,atat,awt;
        printf("Enter the number of processes : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i]=i+1;
                printf("\nArrival time of Process P%d :",p[i]);
                scanf("%d",&at[i]);
                printf("\nBurst time of Process P%d :",p[i]);
                scanf("%d",&bt[i]);
        }
        printf("\nProcess Id\tArrival Time\tBurst Time\n");
        for(i=0;i<n;i++)
        {
                printf("P%d\t\t%d\t\t%d\n",p[i],at[i],bt[i]);
        }
        for(j=1;j<n;j++)
       {
          for(k=0;k<n-j;k++)
           {
             if(at[k]>at[k+1])
             {
               swap(&p[k],&p[k+1]);
              swap(&at[k],&at[k+1]);
              swap(&bt[k],&bt[k+1]);
             }
          }
     }
for(i=0;i<n;i++)
{
        sum = sum + bt[i];
        ct[i] = ct[i] + sum;

        tat[i] = ct[i] - at[i];
        ttat = ttat + tat[i];
```

Signature Not Verified

Principal.VJEC

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

Memory Allocation Methods

Aim

Write a program to implement memory allocation methods for fixed partition
a) First Fit
b) Worst fit
c) Best fit

Algorithm for Main()

1. Start
2. Read the no of blocks and Process as 'bno' and 'Pno' and read the size of block and Process as bsize[] and Psize[] using for loop
3. Repeat the following steps till Exit function is called
   3.1 Read the choice as ch
   3.2 If ch = 1, call the function firstfit with Parameter two bno, bsize, Pno and Psize, breaks
   3.3 If ch = 2, call the function bestfit with Parameter bno, size, Pno and Psize, breaks
   3.4 If ch = 3, call the function worstfit with Parameter bno, bsize, Pno and Psize, breaks
   3.5 otherwise call the Exit function

4. stop

Algorithm for firstfit

1. start
2. Declare allocation [50], i, j
3. Set i=0, set allocation [i] = -1 till i<b, i = i+1
4. Set i=0, repeat the following tell i<p
   4.1 Set j=0, repeat the following tell j<b
      4.1.1 check if allocation [j] = -1 and bs [j] >= ps [i]
         4.1.1.1 if Yes, allocation [j] = i, break
5. Set i=0 repeat the following till i<b
   5.1 Print i+1, bs [i]
   5.2 check if allocation [i] != -1 if yes
      5.2.1 Print allocation [i]+1, ps [allocation [i]]
      5.2.2 otherwise "Print not allocated
   5.3 i = i+1
6. Stop

Algorithm for bestfit

1. start
2. Declare allocation [50], i, j, bi
3. Set i=0, set allocation [i] = -1 tell i<b, i=i+1

4. Set i=0, set allocation[i] Repeat the following till i<p

4.1 Set bi = -1

4.2 set j=0, repeat the following till j<b

4.2.1 check if allocation[j] = -1 and bs[j] >= ps[i], if yes

4.2.1.1 check if bf=1, if yes

a. bf=j, else if check bs[f] > bs[j]

b. if yes bi=j

4.2.2 checks if bf ≠ -1 if yes, set allocation[bi] = f

5. Print 'Block no, block size, Process No, Process Size'

6. Set f=0, repeat the following till f<b

6.1 Print f+1, bs[i]

6.2 if allocation[i] != -i, if yes;

6.2.1 Print allocation[i] +1, ps[allocation[i]]

6.2.2 otherwise Print 'not allocated'

6.3 i = f+1

7. stop

Algorithm for worst fit

1. Start

2. Declare allocation[sp], i, j, b

3. Set f=0, set allocation[i] = -1 till f<b, i = i+1

4. Set i=0 repeat the following till i<p
   4.1 bi = -1
   4.2 Set j=0, repeat the following till j<b
   4.2.1 if allocation[j] = -1 and bs[j] >= ps[i],
   if yes
   4.2.1.1 check if bf = -1, then bf = j
   4.2.1.2 otherwise check if bs[bi] < bs[j]
   then bf = j
   4.3 j = j+1
   4.4 check if bf ≠ -1, then set allocation
   b[i] = f
   4.5 i = i+1
5. Print block no, block size, Process No, Process size
6. set i=0, repeat the following till i<b
   6.1 Print i+1, bs[i]
   6.2 check if allocation[i] ≠ -1 if yes, then Print ps[allocation[i]], otherwise Print "Not allocated"
   6.3 i = i+1
7. stop

Result

The Program has been executed Successfully and the output is obtained

Experiment No : 8
Date : 21/07/2022

sheethal cp

MEMORY ALLOCATION METHODS

```
#include<stdio.h>
#include<stdlib.h>
void firstfit(int b,int bs[],int p,int ps[])
{
        int allocation[50],i,j;
        for(i=0;i<b;i++)
        {
                allocation[i]=-1;
        }
        for(i=0;i<p;i++)
        {
                for(j=0;j<b;j++)
                {
                        if(allocation[j]==-1&& bs[j]>=ps[i])
                        {
                                allocation[j]=i;
                                break;
                        }
                }
        }
        printf("\nblock no.\tblock size\tprocess no.\tprocess size\n");
        for(i=0;i<b;i++)
        {
                printf("\n%d\t\t%d\t\t",i+1,bs[i]);
                if(allocation[i]!=-1)
                {
                        printf("%d\t\t%d",allocation[i]+1,ps[allocation[i]]);
                }
                else
                {
                        printf("not allocated");
                }
        }
}
void bestfit(int b,int bs[],int p,int ps[])
{
        int allocation[50],i,j,bi;
        for(i=0;i<b;i++)
        {
                allocation[i]=-1;
        }
        for(i=0;i<p;i++)
        {
                bi=-1;
                for(j=0;j<b;j++)
                {
```

Signature Not Verified

Principal.VJEC

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						14
Total Marks (45)						44
Signature						

Experiment No: 9

Date : 21/07/2022

Page Replacement algorithm

Aim

Write a Program to implement Page replacement algorithm

a) FIFO

b) LRU

Algorithm

Algorithm for fifo

1. start
2. Declare n, a[50], no, frame[50], i, j, k, avail, pf = 0 of int data type
3. Read the no. of pages as n
4. Read all the page numbers and store in a[i] using for loop
5. Read the number of frames as no
6. set i=0, set fram[i] = -1 till i<no, i = i+1
7. Set j = 0; set j=1, repeat the following till i<=n

7.1 Print a[i], set avail = 0

7.2 Set k=0, repeat the following till k<no

7.2.1 check if frame[k] = a[i], if yes

Set avail = 1

7.2.2 = k+1

7.3 check if avail = 0, if yes

7.3.1 Set frame[j] = a[i]

7.3.2 Set j = (j+1)%.no

7.3.3 pf = pf+1

7.4 Set k=0, Print frame[k] till k<no,

k++

7.5 i = i+1

8. Print the no. of page fault as pf

9. stop

Algorithm

1. Start
2. declare arrays, frame, time with size 50
3. Enter the Parent " enter the page number
4. Print " Enter the frames
5. Print Reference string
6. Set i=1 and repeat the following till i<=n

6.1 Print a[i]

6.2 Set flag1 = 0 and flag2 = 0

6.3 Set j=0 and repeat the following till j<no

6.4 check if frame[j] = a[i]

6.4.1 Set counter++

6.4.2 set [j] = counter

6.4.3 increment pf

8.4.4 set flag2 = 1

7. Set if (flag2 == 0)

7.1 Set pos = FindLRU with parameter time & no

7.2 Set frame[pos] = a[i]

7.3 counter increment counter

7.4 Set time[pos] = counter

7.5 Increment pf

8. Set k=0, repeat till k<no

8.1 Print frame

9. Print number of Page fault

10. Stop

Algorithm for Find(LRU C)

1. Start

2. declare i, min and p=0

3. Set i=0 repeat till i<1

4. check if t[i] < min if true set min = t[i]

5. Step p=i and return p

6. Stop.

Result

The Program has been executed and Verified output

10/5/2022

Signature Not Verified

Principal.VJEC

PAGE REPLACEMENT ALGORITHM

FIFO

```
#include<stdio.h>
int main()
{
        int n,a[50],no,frame[50],i,j,avail,k,pf=0;
        printf("enter the no.of pages\n");
        scanf("%d",&n);
        printf("enter page number\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("enter the no.of frames\n");
        scanf("%d",&no);
        for(i=0;i<no;i++)
        {
                frame[i]=-1;
        }
        j=0;
        printf("Referencre string\tpage frame\n");
        for(i=1;i<=n;i++)
        {
                printf("%d\t\t",a[i]);
                avail=0;
                for(k=0;k<no;k++)
                {
                        if(frame[k]==a[i])
                        {
                                avail=1;
                        }
                }
                if(avail==0)
                {
                        frame[j]=a[i];
                        j=(j+1)%no;
                        pf++;
                }
                for(k=0;k<no;k++)
                {
                        printf("%d\t",frame[k]);
                }
                printf("\n");
        }
        printf("Number of page fault=%d\n",pf);
        return 0;
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Signature Not Verified

Principal.VJEC

Experiment No: 10
Date : 25/07/2022

64

Bankers Algorithm

Aim

Write a program to implement the Bankers algorithm for deadlock avoidance

Algorithm

Available : It is a 1D array of size 'm' indicating the no. of available resource of each type
Available[j] = k means there are 'k' instances of resource type Rj

MAX: It is 2D array of size 'n x m', that defines the maximum demand of each Process in a System. MAX[i,j] = k means Process Pi may request at most 'k' instance of resource type Rj

Allocation: It is a 2D array of size 'n x m' that defines the no. of resources of each type currently allocated to each Process
Allocation[i,j] = k means Process Pi is currently allocated 'k' instances of resource type Rj

Need: It is 2D array of size 'n x m' that indicates the remaining resource need of each process. Need[i,j] = MAX[i,j] - Allocation[i,j]

Algorithm

Step: 1 Let work and Finish be vectors of length 'm' and 'n' respectively

Initialize : Work = Available

Finish[i] = False for i = 1, 2, 3, 4 .... n

2. Find an i such that both

a) Finish [i] = False

b) Need i <= Work, then go to step 3 if no such 'i' exist goto step 4

3. Work = Work + Allocation [i]

Finish [i] = True

goto Step 2

4. If Finish [i] = True for all i, then the system is in safe state, other wise in waiting state

Result

Navya 26/09/22

Programs has Executed and output has been verified

Experiment No. : 10
Date : 25-07-2022

BANKER'S ALGORITHM

```
#include<stdio.h>
void main()
{
        int p,r,allocation[20][20],max[20][20],available[20],need[20]
[20],i,j,work[20],finish[20],counter,s[20],l,t,flag;
        printf("Enter the number of processes : ");
        scanf("%d",&p);
        printf("\nEnter the number of resources : ");
        scanf("%d",&r);
        printf("\nEnter Allocation Matrix : ");
        for(i=0;i<p;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&allocation[i][j]);
                }
        }
        printf("\nAllocation Matrix : ");
        for(i=0;i<p;i++)
        {
                for(j=0;j<r;j++)
                {
                        printf("%d\t",allocation[i][j]);
                }
                printf("\n");
        }
        printf("\nEnter Max Matrix : ");
        for(i=0;i<p;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&max[i][j]);
                }
        }
        printf("\nMax Matrix : ");
        for(i=0;i<p;i++)
        {
                for(j=0;j<r;j++)
                {
                        printf("%d\t",max[i][j]);
                }
                printf("\n");
        }
        printf("\nEnter Available Matrix : ");
        for(i=0;i<r;i++)
```

RUBRICS TO EVALUATE PROGRAMMING EXPERIMENTS

No	Performance criteria	Excellent - 6	Good - 4	Satisfactory -2	Poor -1	Total
1	Algorithm	The algorithm is of • Minimum no. of steps (Optimal) • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Steps should be computable • Sequentially executable	The algorithm is • Logically correct • Sequentially executable	The algorithm is • Logically correct only	6
2	Program	The code is exceptionally well organized and very easy to follow and has a good logic.	The code is fairly easy to read.	The code is readable only by someone who knows it well.	The code is poorly organized and so difficult to read.	6
3	Output	Program gives expected result for all possible inputs.	Program gives expected result for most of the inputs.	Program produces correct results but not in specified format.	Program produces incorrect results.	6
4	Time taken	The program was completed within 1 hour.	The program was completed within 90 minutes.	The program was completed within the lab session.	Took more than one lab session to complete.	6
5	Record	Neat, well labelled, organized and completed within time.	Neat, work is somewhat organized and completed.	Somewhat organized but not neat.	Not organized and not neat. Some entries are incomplete.	6
Score for this experiment (30)						30
Viva (15)						15
Total Marks (45)						45
Signature						

Experiment NO: 14

Date : 25/07/2022

Disk Scheduling Algorithm

Aim

write a Program to Simulate following disk Scheduling algorithm

a. FCFS

b. SCAN

c. C-SCAN

Algorithm for FCFS

Step 1 : start

Step 2 : declare array x with size 100

3 : Enter the number of request 'n'

4 : Enter the sequence of request by using for loop

5 : Enter the initial head position 'h'

6 : Print the head position h

7 : Set i=0 and repeat the following steps till i<n

7.1 Print "→" and x[i]

7.2 Set t = t + abs (x[i] - h)

7.3 Set h = x[i]

8 : Print the total number of head movement 't'

9 Stop

Algorithm for SCAN

1. Start
2. declare an array 'x' with size = 100
3. Read the number of request as 'n'
4. Read the sequence of requests by using for loop
5. read the initial head position as 'h'
6. read the no. of cylepairs as 's'
7. Set i = 1 and repeat the following steps till i < n
   - 7.1 Set j = 0 repeat the following steps till j < n-1
   - 7.2 check if x[j] > x[j+1] if true goto next step
     - 7.2.1 temp = x[j]
     - 7.2.2 x[j] = x[j+1]
     - 7.2.3 x[j+1] = temp
   - 7.3 j = j+1
8. i = i+1
9. Set i = 0 and ~~repeat~~ the following step till i < n
   - 9.1 check if h < x[i] if true goto next step
     - 9.1.1 Set index = i
     - 9.1.2 break the loop

Step 10: i = i+1

11. Set i = index and repeat the steps till i < n
    - 11.1 Print "→" and x[i]
    - 11.2 Set t = t + abs [x[i] - h]

11.3 $h = s[i]$

12. $i = i+1$

13. Set $t = t + abs[(s-1) - s(i-1)]$

14. $h = s-1$

15. Print "→" and "h"

16. Set i = index and repeat the following steps till $i >= 0$

16.1 Print "→" and $s[i]$

16.2 Set $t = t + abs[s[i] - h]$

16.3 $h = s[i]$

17. $i = i - t$

18. Print "total number of head movement" : t

19. stop

Algorithm for C-SCAN

1. Start

2. Read the no. of request sequence of request, initial head position and no. of cylinders

3. Set $i=1$ and repeat the following till $i<n$

3.1 Set $j=0$, Repeat the following till $i<n$

3.1.1 check if $s[j] > s[i+1]$ if yes goto (a)

a) Set temp = $s[j]$

b) Set $s[j] = s[i+1]$

c) Set $s[i+1]$ = temp

3.1.2 Set j = j+1

3.2 set i = i+1

4. set i = 0, Repeat the following till i<n

4.1 check of ~~k~~ x[i]

4.1.1 Set index = i then break

4.2 Set i = i+1

5. Print head position

6. Set i = index and repeat the following till i<n

6.1 Print x[i]

6.2 Set t = t+abs (x[i] - h)

6.3 Set h = x[i]

7. Set t = t+abs ((s-1) + x[i-1])

8. Print (s-1)

9. Set t = t+abs (s-1) and h = 0

10. Print head value

11. Set i=0. Repeat the following till i<index

11.1 Print x[i]

11.2 Set t = t+abs (x[i] - h)

11.3 set h = x[i]

12. Print Total no. of head movements

13. Stop

Result

The Program has been Executed Successfully and Verified the output

Nayana 26/04/2022

Name : Sheethal C P
Roll No. : 51

Experiment No. : 11
Date : 25-07-2022

DISK SCHEDULING ALGORITHMS

1. Program for FCFS

```
#include<stdio.h>
#include<math.h>
void main()
{
        int r[100],n,i,t=0,h;
        printf("\nEnter the number of requests : ");
        scanf("%d",&n);
        printf("\nEnter sequence of requests : ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&r[i]);
        }
        printf("\nEnter initial head position : ");
        scanf("%d",&h);
        printf("%d",h);
        for(i=0;i<n;i++)
        {
                printf("-> %d",r[i]);
                t=t+abs(r[i]-h);
                h=r[i];
        }
        printf("\nTotal number of head movements = %d\n",t);
}
```

Output

```
pglab@pglab-H310M-H:~/Desktop/VML20CS147/os-lab$ gcc diskfcfs.c -o diskfcfs.out -lm
pglab@pglab-H310M-H:~/Desktop/VML20CS147/os-lab$ ./diskfcfs.out

Enter the number of requests : 8

Enter sequence of requests : 98 183 41 122 14 124 65 67

Enter initial head position : 53

53-> 98-> 183-> 41-> 122-> 14-> 124-> 65-> 67

Total number of head movements = 632
```



```
        printf("-> %d",h);
        printf("-> %d",h);
```