1. **a) Program to count the number of characters, words, spaces and lines in a given input file.**

```
%{
        #include<stdio.h>
        int cc=0,wc=0,sc=0,lc=0;
%}
%%
[\n]            {lc++;cc++;}
[^ \t\n]+       {wc++,cc+=yyleng;}
" "             {sc++;cc++;}
.               {cc++;}
%%
main(int argc,char * argv[])
{
        if(argc>1)
        {
                FILE *f = fopen(argv[1],"r");
                if(!f)
                {
                        printf("Error openning in file");
                        exit(1);
                }
                yyin=f;
        }
        yylex();
        printf("Line        count=%d\nWord        count=%d\nChar        count=%d\nSpace
count=%d\n",lc,wc,cc,sc);
}
```

b) Program to count the numbers of comment lines in a given C program. Also eliminate them and copy the resulting program into separate file.

```
%{
        #include<stdio.h>
        int sc=0,mc=0;
%}
%x cmnt
%%
"//".* {sc++;}
"/*" {BEGIN cmnt;mc++;}
<cmnt>.         ;
<cmnt>\n;
<cmnt>"*/" {BEGIN 0;}
%%
main (int argc,char *argv[])
{
        if(argc!=3)
        {
                printf("No files");
                exit(0);
        }

        yyin=fopen(argv[1],"r");

        yyout=fopen(argv[2],"w");

        yylex();

        printf("Single line comment count:%d\n",sc);

        printf("Multi line comment count:%d\n",mc);

        printf("Total comment count:%d\n",sc+mc);

}
```

2. a) Program to recognize a valid arithmetic expression and to recognize the identifiers and operators present. Print them separately.

```
%{
        #include <stdio.h>
        int count=0;
        char ch;
%}

%%
"int "|"char "|"float "|"double " { while(1)
        {
                ch=input();
                if(ch==',')
                count++;
                else if(ch==';')
                {
                        count++;
                        break;
                }
        }
}
.|\n ;
%%
int main(int argc,char *argv[])
{
        if(argc!=2)
        {
                printf("Invalid number of arguments!\n");
                return 1;
        }
        yyin=fopen(argv[1],"r");
        yylex();
        printf("Number of identifiers : %d\n",count);
        return 0;
}
```

b) Program to recognize whether a given sentence is simple or compound.

```
%{
        #include<stdio.h>
        int f=1;
%}
id [a-zA-Z0-9]
%%
{id}" and "|" or "|" but "|" because "{id}      {f=0;}
.                                                ;
\n                                               ;
%%
main()
{
        printf("Enter a sentence(CTRL+D to exit):");
        yylex();
        if(f==1)
                printf("It is a Simple sentence\n");
        else
                printf("It is a Compound sentence\n");
}
```

3. Program to recognize and count the number of identifiers in a given input file. Design, develop, and execute the following programs using YACC:

```
%{
        #include <stdio.h>
        int count=0;
        char ch;
%}
%%
"int "|"char "|"float "|"double " { while(1)
{
        ch=input();
        if(ch==',')
        count++;
        else if(ch==';')
        {
                count++;
                break;
        }
}
}
.|\n ;
```

```
%%

int main(int argc,char *argv[])
{
	if(argc!=2)
	{
		printf("Invalid number of arguments!\n");
		return 1;
	}
	yyin=fopen(argv[1],"r");
	yylex();
	printf("Number of identifiers : %d\n",count);
	return 0;
}
```

4. a) Program to recognize a valid arithmetic expression that uses operators +, -, * and /.

lex

```
%{
/* 4a.l */
#include "y.tab.h"
%}
%%
[0-9]+(\.[0-9]+)?        { return NUM;}
[a-zA-Z][_a-zA-Z0-9]*          { return ID; }
[\t]                     ;
\n                       return 0;
.                        return yytext[0];
%%
```

yacc

```
%{
#include<stdio.h>
%}
%token NUM ID
%left '+' '-'
%left '*' '/'
%%
e : e '+' e
|  e '-' e
|  e '*' e
|  e '/' e
|  '('e')'
| NUM
| ID        ;
%%
```

```
main()
{
printf(" Type the Expression & Press Enter key\n");
yyparse();
printf(" Valid Expression \n");
}
yyerror()
{
printf(" Invalid Expresion!!!!\n");
exit(0);
}
```

b) Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

<u>lex</u>

```
%{
        #include "y.tab.h"
%}
%%
[ \t]   ;
[0-9]+          {return DIGIT;}
[a-zA-Z]+       {return ALPHA;}
.               {return yytext[0];}
\n              {return 0;}
%%
```

<u>yacc</u>

```
%{
        #include<stdio.h>
        int f=1;
%}
%token          ALPHA DIGIT
%%
VAR:ALPHA|VAR ALPHA|VAR DIGIT ;
```

```
%%
yyerror()
{
	f=0;
}
main()
{
	printf("Enter a variable:\n");
	yyparse();
	if(f)
		printf("\nValid variable name");
	else
		printf("\nInvalid variable name");
}
```

5. a) Program to evaluate an arithmetic expression involving operators +, -, * and /.

lex

```
%{
/* 5a.l */
#include<stdlib.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+          {  yylval=atoi(yytext);     return NUM;    }
[\t]         ;
\n           return 0;
.            return yytext[0];
%%
```

yacc

```
%{
#include <stdio.h>
%}
%token NUM
%left '+' '-'
%left '*' '/'
%%
expr : e { printf(" Result : %d\n",$1); return 0;};
e : e '+' e {$$=$1+$3;}
|  e '-' e {$$=$1-$3;}
|  e '*' e {$$=$1*$3;}
|  e '/' e {$$=$1/$3;}
|  '('e')' {$$=$2;}
| NUM      {$$=$1;};
```

```
%%
main()
{
printf(" Type the Expression & Press Enter key\n");  yyparse();
printf(" Valid Expression \n");
}
yyerror()
{
 printf(" Invalid Expresion!!!!\n");  exit(0);
}
```

b) Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using the grammar (anbn, n>= 0).

<u>lex</u>

```
%{
        #include"y.tab.h"
%}
%%
a       return A;
b       return B;
[0-9]   {yylval=atoi(yytext);return NUM;}
\t      ;
.       return yytext[0];
\n      return 0;
%%
```

<u>yacc</u>

```
%{
        #include<stdio.h>
        int na=0,nb=0,f=1;
%}
%token A B NUM
```

```
%%
VAR:   A       {na++;}
        |B       {nb++;}
        |VAR A         {na++;}
        |VAR B         {nb++;}
        |NUM  {f=0;}
        |VAR NUM {f=0;}
  ;
%%
main()
{
        yyparse();
        if(na==nb && f==1)
                printf("Valid");
        else
                printf("Invalid");
}
```

6. Program to recognize the grammar (anb, n>= 10).

<u>lex</u>

```
%{
        #include"y.tab.h"
%}
%%
a       return A;
b       return B;
[0-9]   {yylval=atoi(yytext);return NUM;}
\t      ;
.       return yytext[0];
\n      return 0;
%%
```

<u>yacc</u>

```
%{
        #include<stdio.h>
        int na=0,nb=0,f=1;
%}
%token A B NUM
%%
VAR:   A        {na++;}
        |B        {nb++;}
        |VAR A         {na++;}
        |VAR B         {nb++;}
        |NUM  {f=0;}
        |VAR NUM {f=0;}
  ;
%%
main()
```

```
{
	yyparse();
	if(na>=10 && nb==1 && f==1)
		printf("Valid");
	else
		printf("Invalid");
}
```

7. **a) Non-recursive shell script that accepts any number of arguments and prints them in the Reverse order, (For example, if the script is named rargs, then executing rargs A B C should produce C B A on the standard output).**

```
#!/bin/bash
rev=""
if [ $# -eq 0 ];
then
        echo "Usage error"
        exit
fi
echo "Total no of args: $#"
echo "Args are: $*"
echo "Args in reverse: "
for i in $*
do
        rev=$i" "$rev
done
echo $rev
```

b) C program that creates a child process to read commands from the standard input and execute them (a minimal implementation of a shell – like program). You can assume that no arguments will be passed to the commands to be executed.

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

        pid_t cpid,ppid,pid;

        if((pid=fork())<0)

        {

                printf("Fork error");

                exit(0);

        }

        if(pid==0)

        {

                printf("\n I am Child");

                printf("\n My id is %d", getpid());

                printf("\n My parent id is %d", getppid());

                exit(0);

        }
```

```
	else
	{
		wait();
		printf("\n I am Parent");
		printf("\n My id is %d", getpid());
		printf("\n My child id is %d\n", pid);
	}
}
```

8. a) Shell script that accepts two file names as arguments, checks if the permissions for these files are identical and if the permissions are identical, outputs the common permissions, otherwise outputs each file name followed by its permissions.

```
if [ $# -ne 2 ];
then
        echo "Error"
        exit
fi
perm1=`ls -l $1 | cut -c 1-10`
perm2=`ls -l $2 | cut -c 1-10`
u1=`ls -l $1 | cut -c 2-4`
u2=`ls -l $2 | cut -c 2-4`
g1=`ls -l $1 | cut -c 5-7`
g2=`ls -l $2 | cut -c 5-7`
o1=`ls -l $1 | cut -c 8-10`
o2=`ls -l $2 | cut -c 8-10`
r1=`ls -l $1 | cut -c 2,5,8`
r2=`ls -l $2 | cut -c 2,5,8`
w1=`ls -l $1 | cut -c 3,6,9`
w2=`ls -l $2 | cut -c 3,6,9`
x1=`ls -l $1 | cut -c 4,7,10`
x2=`ls -l $2 | cut -c 4,7,10`
if [ $perm1 = $perm2 ];
then
        echo "Same file permissions"
        echo "Permissions of $1 and $2 are $perm1"
else
        echo "Different file permissions"
        if [ $u1 = $u2 ];
        then
        echo "Same user permissions"
        echo "File 1 $1 and $2: $u1"
else
        echo "Different user permissions"
        echo "File 1 $1 : $u1"
        echo "File 1 $2 : $u2"
fi
if [ $g1 = $g2 ];
then
        echo "Same group permissions"
        echo "File 1 $1 and $2: $g1"
else
        echo "Different group permissions"
        echo "File 1 $1 : $g1"
        echo "File 2 $2 : $g2"
fi
if [ $o1 = $o2 ];
then
        echo "Same user permissions"
        echo "File 1 $1 and $2: $o1"
else
        echo "Different world permissions"
        echo "File 1 $1 : $o1"
        echo "File 2 $2 : $o2"
fi
if [ $r1 = $r2 ];
then
        echo "Same read permissions"
        echo "File 1 $1 and $2: $r1"
else
        echo "Different read permissions"
```

```
                echo "File 1 $1 : $r1"
                echo "File 1 $2 : $r2"
        fi
        if [ $w1 = $w2 ];
        then
                echo "Same group permissions"
                echo "File 1 $1 and $2: $w1"
        else
                echo "Different write permissions"
                echo "File 1 $1 : $w1"
                echo "File 2 $2 : $w2"
        fi
        if [ $x1 = $x2 ];
        then
                echo "Same execute permissions"
                echo "File 1 $1 and $2: $x1"
        else
                echo "Different execute permissions"
                echo "File 1 $1 : $x1"
                echo "File 2 $2 : $x2"
        fi
fi
```

b) C program to create a file with 16 bytes of arbitrary data from the beginning and another 16 bytes of arbitrary data from an offset of 48. Display the file contents to demonstrate how the hole in file is handled.

```
#include<stdio.h>

#include<fcntl.h>

main()

{

        char buf1[]="abcdefghijklmnop";

        char buf2[]="ABCDEFGHIJKLMNOP";

        int fd = open("K1",O_RDWR|O_CREAT|O_EXCL,0755);

        write(fd,buf1,16);

        lseek(fd,32,SEEK_CUR);

        write(fd,buf2,16);

        close(fd);

        system("vi K1");

}
```

9. a) Shell script that accepts file names specified as arguments and creates a shell script that contains this file as well as the code to recreate these files. Thus if the script generated by your script is executed, it would recreate the original files(This is same as the "bundle" script described by Brain W. Kernighan and Rob Pike in " The Unix Programming Environment", Prentice – Hall India).

```
#!/bin/bash
echo "#to unbundle,bash this file"
for i
do
echo "echo $i 1>&2"
echo "cat >$i << 'End of $i'"
cat $i
echo "End of $i"
done
```

b) C program to do the following: Using fork( ) create a child process. The child process prints its own process-id and id of its parent and then exits. The parent process waits for its child to finish (by executing the wait( )) and prints its own process-id and the id of its child process and then exits.

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

        pid_t cpid,ppid,pid;

        if((pid=fork())<0)

        {

                printf("Fork error");

                exit(0);

        }

        if(pid==0)

        {

                printf("\n I am Child");

                printf("\n My id is %d", getpid());

                printf("\n My parent id is %d", getppid());

                exit(0);
        }
        else
        {
                wait();
                printf("\n I am Parent");
                printf("\n My id is %d", getpid());
                printf("\n My child id is %d\n", pid);
        }
}
```

10. Design, develop and execute a program in C / C++ to simulate the working of Shortest Remaining Time and Round-Robin Scheduling Algorithms. Experiment with different quantum sizes for the Round- Robin algorithm. In all cases, determine the average turn-around time. The input can be read from key board or from a file.

```
#include<stdio.h>

typedef struct
{
        int name;
        int at;
        int burst;
        int wait;
        int tat;
        int remaining;
        int flag;
}process;
process p[5], p1[5];
int n, n1;
void round_robin();
void accept();
void sjf();
int minimum();
int tq, t, i, c=0, rtwt=0, rttat=0, temp, sq=0, stwt=0, sttat=0;
float rawt=0.0, ratat=0.0, sawt=0.0, satat=0.0;
void main()
{
        int ch;
        printf("\nChoose one of the folowing options:");
        printf("\n\t1-SJF\n\t2-Round Robin\n\tExit");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                                accept();
                                sjf();
                                break;
                case 2:
                                round_robin();
                                break;
                case 3:
                                break;
                default:
                                printf("\nInvalid Entry");
        }
}
void accept()
{
        int i;
        printf("\nEnter the total number of procedures: ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter the procedure name: ");
                scanf("%d",&p[i].name);
                printf("Enter the arrival time: ");
                scanf("%d",&p[i].at);
                printf("Enter the burst time: ");
                scanf("%d",&p[i].burst);
                p[i].wait=0;
        }
```

```
}
void sjf()
{
        int i, j, tbt=0, no, k;
        for(i=0;i<n;i++)
        {
                p[i].remaining=p[i].burst;
                tbt=tbt+p[i].burst;
        }
        printf("\n");
        for(i=0;i<tbt;i++)
        {
                for(j=0;j<n;j++)
                {
                        p[j].flag=0;
                        if(p[j].at<=i && p[j].remaining>0)
                        {
                                p[j].flag=1;
                        }
                }
                no=minimum();
                printf("%d-P%d-",i,no);
                p[no].remaining=p[no].remaining-1;

                for(k=0;k<n;k++)
                {
                        if(p[k].at<=i && p[k].remaining>0 && p[k].name!=no)
                        {
                                p[k].wait=p[k].wait+1;
                        }
                }
```

```
	}
	printf("%d",i);

	for(k=0;k<n;k++)
		p[k].tat=p[k].burst+p[k].wait;

	printf("\n\nProcess Name\tArrival\tBurst\tWait\tTurn Around Time");
	for(k=0;k<n;k++)
	{
		stwt=stwt+p[k].wait;
		sttat=sttat+p[k].tat;
		printf("\n          P%d\t\t          %d\t%d\t%d\t\t%d",   p[k].name,   p[k].at,
p[k].burst,p[k].wait,p[k].tat);
	}
	sawt=(float)stwt/n;
	satat=(float)sttat/n;
	printf("\nstwt=%d\nsttat=%d\nsatwt=%f\nsatat=%f\n\n", stwt, sttat, sawt, satat);
}
int minimum()
{
	int i, j, min, pro_no, k;
	min=999;
	for(i=0;i<n;i++)
	{
		if(p[i].flag==1)
		{
			if(p[i].remaining<min)
			{
				min=p[i].remaining;
				pro_no=i;
			}
```

```
                }
        }
        return(pro_no);
}
void round_robin()
{
        printf("\nEnter the number of processes: ");
        scanf("%d",&n1);
        printf("\nEnter the burst time for sequences:\n");
        for(i=0; i<n1; i++)
        {
                scanf("%d",&p1[i].burst);
                p1[i].remaining=p1[i].burst;
                p1[i].at=0;
        }
        printf("\nEnter the time quantum: ");
        scanf("%d",&tq);
        printf("\n");
        t=0;
        while(1)
        {
                for(i=0,c=0;i<n1;i++)
                {
                        temp=tq;
                        if(p1[i].remaining==0)
                        {
                                c++;
                                continue;
                        }
                        if(p1[i].remaining>tq)
                        {
```

```
                    p1[i].remaining=p1[i].remaining-tq;
                    printf("%d-",t);
                    t=t+tq;
                    printf("P%d-",i);
                }
                else if(p1[i].remaining>=0)
                {
                    temp=p1[i].remaining;
                    printf("%d-",t);
                    t=t+p1[i].remaining;
                    p1[i].remaining=0;
                    printf("P%d-",i);
                }
                sq=sq+temp;
                p1[i].tat=sq;
        }
        if(n1==c)
                break;
}
printf("%d",t);
for(i=0;i<n1;i++)
{
        p1[i].wait=p1[i].tat-p1[i].burst;
        rtwt=rtwt+p1[i].wait;
        rttat=rttat+p1[i].tat;
}
rawt=(float)rtwt/n1;
ratat=(float)rttat/n1;
printf("\n\nProcess_no\tBurst time\tWait Time\tTurn Around Time\n");
for(i=0;i<n1;i++)
        printf("P%d\t\t%d\t\t%d\t\t%d\n",i+1,p1[i].burst,p1[i].wait,p1[i].tat);
        printf("\nAverage waiting time is: %f",rawt);
        printf("\nAverage turn around time is: %f\n\n",ratat);
}
```

11. Using OpenMP, Design, develop and run a multi-threaded program to generate and print Fibonacci Series. One thread has to generate the numbers up to the specified limit and another thread has to print them. Ensure proper synchronization.

```
#include<stdio.h>
#include<omp.h>
int main()
{
        int a[100],i,n;
        omp_set_num_threads(2);

        printf("\n\tEnter the no. of terms:");
        scanf("%d",&n);

        a[0]=0;
        a[1]=1;
        #pragma omp parallel
        {
                #pragma omp single
                for(i=2;i<n;i++)
                {
                        a[i]=a[i-2]+a[i-1];
                        printf("\n\tId of thread:%d Number
generated:%d",omp_get_thread_num(),a[i]);
                }
                #pragma omp barrier
                #pragma omp single
                {
                        printf("\n\n\tFibonacci series generated:\n");
                        for(i=0;i<n;i++)
                        {
```

```
                                        printf("\tNo:%d                                          Thread:%d
Value:%d\n",i+1,omp_get_thread_num(),a[i]);
                              }
                    }
          }
          printf("\n");
}
```

12. Design, develop and run a program to implement the Banker's Algorithm. Demonstrate its working with different data values.

```
#include<stdio.h>
int main()
{
	int clm[7][5],req[7][5],alloc[7][5],rsrc[5],avail[5],comp[7],work[5],flag[7],safe[7];
	int p,r,i,j,c,t,k,x;
	c=0;
	printf("\n\tEnter the no. of processes:");
	scanf("%d",&p);
	for(i=0;i<p;i++)
		comp[i]=0;
	printf("\tNo. of resources:");
	scanf("%d",&r);
	printf("\n\tClaim:\n");
	for(i=0;i<p;i++)
	{
		printf("\tProcess %d:",i);
		for(j=0;j<r;j++)
			scanf("%d",&clm[i][j]);
	}
	printf("\n\tAllocation for each process:\n");
	for(i=0;i<p;i++)
	{
		printf("\tProcess %d:",i);
		for(j=0;j<r;j++)
			scanf("%d",&alloc[i][j]);
	}
	printf("\n\tTotal no. of resources:");
	for(j=0;j<r;j++)
```

```
        scanf("%d",&rsrc[j]);
for(j=0;j<r;j++)
{
        int tot=0;
        avail[j]=0;
        for(i=0;i<p;i++)
                tot+=alloc[i][j];
        avail[j]=rsrc[j]-tot;
}
for(i=0;i<p;i++)
        for(j=0;j<r;j++)
                req[i][j]=clm[i][j]-alloc[i][j];
printf("\n\tAvailable resources:");
for(j=0;j<r;j++)
{
        work[j]=avail[j];
        printf("%d ",avail[j]);
}
printf("\n\n\t:Claim:\t\t\t:Allocation:\t\t:Need:\n\t  ");
for(i=0;i<p;i++)
{
        for(j=0;j<r;j++)
                printf("%d",clm[i][j]);
        printf("\t\t\t");
        for(j=0;j<r;j++)
                printf("%d",alloc[i][j]);
        printf("\t\t\t");
        for(j=0;j<r;j++)
                printf("%d",req[i][j]);
        printf("\n\t  ");
}
```

```
k=0;
x=0;
do
{
        k++;
        for(i=0;i<p;i++)
                flag[i]=0;
        for(i=0;i<p;i++)
        {
                if(comp[i]==0)
                {
                        for(j=0;j<r;j++)
                                if(req[i][j]>work[j])
                                {
                                        flag[i]=1;
                                        break;
                                }
                        if(flag[i]==0)
                        {
                                for(j=0;j<r;j++)
                                {
                                        work[j]=work[j]+alloc[i][j];
                                        alloc[i][j]=0;
                                        clm[i][j]=0;
                                }
                        comp[i]=1;
                        ++c;
                        safe[x]=i;
                        printf("\n\tSafe=%d",safe[x]);
                        x++;
                        }
                    }
                }
            }
            while(c!=p && k<p);
            printf("\n\n\tk=%d\tCount=%d\n",k,c);
            if(c==p)
            {
                printf("\n\tSystem is in safe state with sequence:");
                for(j=0;j<p;j++)
                    printf("P%d ",safe[j]);
                printf("\n\n");
            }
            else
                printf("\n\tSystem is in unsafe state\n\n");
}
```