1

/* Lex program to recognize and count the number of identifiers in a given input */

```
/* Definition Section */
%{
        #include<stdio.h>
        int others = 0;
        int keyword = 0;
        int id = 0;
%}

/*Rules Section*/
%%

"int"|"double"|"float"|"char"|"do"|"while"|"main"|"return"|"printf"|"scanf"|"include"|"stdio.h" {
keyword++;
fprintf(yyout, "%s is keyword\n", yytext);
}

[A-Za-z_][a-zA-Z0-9_]* {
id++;
fprintf(yyout, "%s is identifier\n", yytext);
}

. {others++;}

%%

/* Main Section */
int main()
{
        /* yyin and yyout as pointer of File type */
        extern FILE *yyin, *yyout;

        /* yyin points to the file input.c and opens it in read mode*/
        yyin = fopen("input.c", "r");

        /* yyout points to the file output.txt and opens it in write mode*/
        yyout = fopen("output.txt", "w");

        yylex();
```

```
        fprintf(yyout, "Number of identifier is %d\n", id);
        fprintf(yyout, "Number of keywords is %d\n", keyword);
        return 0;
}
int yywrap()
{
        return 1;
}
```

2

/* Lex program to count the characters, words, spaces and lines in a given input */

```
/*Definition Section*/
%{
        #include<stdio.h>
        int ch=0;
        int word=0;
        int space=0;
        int line=0;
        int others = 0;
%}

/*Rules Section*/
%%

[a-zA-Z] {
        ch++;
        fprintf(yyout, "%s is a character\n", yytext);
}

[a-zA-Z]+ {
        word++;
        fprintf(yyout, "%s is a word\n", yytext);
}

" " {
        space++;
        fprintf(yyout, "%s space\n", yytext);}

'\n' {
```

```
        line++;
        fprintf(yyout, "%s new line\n", yytext);}

%%

/*Main Section*/
int main()
{
        yyin = fopen("input.c","r");
        yyout = fopen("output.txt","w");
        yylex();
        fprintf(yyout, "char is %d,\nword is %d,\nspace is %d,\nline is %d",ch,word,space,line);
        return 0;
}
int yywrap()
{
        return 1;
}
```

3

/* Lex program to count the number of comment lines in a given c/c++/java program */

```
/*definition section*/
%{
        #include<stdio.h>
        int sc=0;
        int mlc=0;
        int others = 0;
        int flag = 0;

%}

/*Rules Section*/
%%

"/*" {
        if(flag==0) flag++;
}
"*/" {
        if(flag==1)
        {
```

```
                mlc++;
                flag=0;
        }
}

[/][/].*"\n" {
        if(flag==0)
        {
                sc++;
                fprintf(yyout, "%s --> is single line comment\n", yytext);
        }
}

.* {others++;}

%%

/*main section*/
int main()
{
        yyin = fopen("input.c","r");
        yyout = fopen("output.txt","w");

        yylex();

        fprintf(yyout, "%d single line comment\n",sc);
        fprintf(yyout, "%d multi line comment\n",mlc);
        return 0;
}
int yywrap()
{
        return 1;
}
```

4

/* Lex program to identify integer, float, exponential number */

```
/*definition part*/
%{
        #include<stdio.h>
```

```
	int in=0;
	int flt=0;
	int expp=0;
	int others=0;
%}

/*Rules Part*/
%%

[+-]?[0-9]+ {
	in++;
	fprintf(yyout, "%s is integer\n", yytext);
}

[+-]?[0-9]*"."[0-9]+ {
	flt++;
	fprintf(yyout, "%s is float\n", yytext);
}

[+-]?[0-9]+[Ee][-+]?[0-9]+ {
	expp++;
	fprintf(yyout, "%s is exponentital\n", yytext);
}

. {others++;}
%%

/*Main part*/
int main()
{
	yyin=fopen("demo.txt","r");
	yyout=fopen("output.txt","w");

	yylex();

	fprintf(yyout, "%d integer number\n", in);
	fprintf(yyout, "%d float number\n", flt);
	fprintf(yyout, "%d exponential number\n", expp);

	return 0;
```

```
}
int yywrap()
{
	return 1;
}
```

5

/*Lex program to identify operator of telephone number*/

```
/*Definition part*/
%{
	#include<stdio.h>
	int others = 0;
%}

/*Rules part*/
%%

"019"[0-9]+ {
	fprintf(yyout, "%s is a banglalink number\n", yytext);
}
"01"[7|3][0-9]+ {
	fprintf(yyout, "%s is a grammen number\n", yytext);
}
"018"[0-9]+ {
	fprintf(yyout, "%s is an robi number\n", yytext);
}
"016"[0-9]+ {
	fprintf(yyout, "%s is an airtel number\n", yytext);
}
[ \n\t] {others++;}
. {others++;}
%%
/*main part*/
int main()
{
	yyin=fopen("demo2.txt","r");
	yyout = fopen("output.txt","w");

	yylex();
	return 0;
}
```

```
int yywrap()
{
        return 1;
}
```

7

/* Lex program to identify whether a given sentence is simple, complex or compound */

```
/* Definition Section */
%{
        #include<stdio.h>
        int flag=0;
%}

/*Rules Section */
%%

and|or|but|if|then|nevertheless  { flag=1; }
although|Although|because|Because|before|Before|even|Even|though|Though|if|If|since|Since|until|Until|when|When { flag=2; }
.  {}
\n { return 0; }

%%

int main()
{
        printf("Enter the sentence:\n");
        yylex();
        if(flag==0)
                printf("Simple sentence\n");
        else if(flag==1)
                printf("compound sentence\n");
        else
                printf("Complex sentence\n");
}

int yywrap( )
{
        return 1;
}
```

// C++ program to calculate the First and Follow sets of a given grammar

```
#include<bits/stdc++.h>
using namespace std;
int cnt, n=0, m=0, k, e;
char calc_first[10][100];   //stores the final result of first sets
char calc_follow[10][100];  //stores the final result of follow sets
char production[10][10];    //stores the production rules
char follow[10], first[10];
char ck;
//Function to calculate follow
void followfirst(char, int, int);
void findfollow(char c);
//Function to calculate first
void findfirst(char c, int q1, int q2)
{
        int j;
        // The case where we encounter a Terminal
        if(!(isupper(c))) {
                first[n++] = c;
        }
        for(j=0;j<cnt;j++)
        {
                if(production[j][0] == c)
                {
                        if(production[j][2] == '#')
                        {
                                if(production[q1][q2] == '\0')  first[n++] = '#';
                                else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                                {
                                        // Recursion to calculate First of New Non-Terminal we
encounter after epsilon
                                        findfirst(production[q1][q2], q1, (q2+1));
                                }
                                else first[n++] = '#';
                        }
                        else if(!isupper(production[j][2]))
                        {
                                first[n++] = production[j][2];
                        }
                        else
```

```
                        {
                                // Recursion to calculate First of New Non-Terminal we encounter
at the beginning
                                findfirst(production[j][2], j, 3);
                        }
                }
        }
}

//main function
int main()
{
   int jm=0, km=0, i, choice;
   char c, ch;
   cnt = 8;

        // The Input grammar
        strcpy(production[0], "E=TR");
        strcpy(production[1], "R=+TR");
        strcpy(production[2], "R=#");
        strcpy(production[3], "T=FY");
        strcpy(production[4], "Y=*FY");
        strcpy(production[5], "Y=#");
        strcpy(production[6], "F=(E)");
        strcpy(production[7], "F=i");

        int kay, ptr=-1;
   char done[cnt];

   //Initializing the calc_first array
   for(k=0;k<cnt;k++){
      for(kay=0;kay<100;kay++){
         calc_first[k][kay] = '!';
      }
   }

   int point1=0, point2, xxx;
   for(k=0;k<cnt;k++)
   {
      c = production[k][0];
```

```
point2 = 0;
xxx = 0;

//checking if first of c has already been calculated
for(kay=0;kay<=ptr;kay++)
{
   if(c == done[kay]) xxx=1;
}

if(xxx==1) continue;

//Function call
findfirst(c, 0, 0);
ptr++;
//Adding c to the calculated list
done[ptr]=c;

cout<<"\n First(";
cout<<c;
cout<<") = { ";
calc_first[point1][point2++] = c;

//printing the first sets of the grammar
for(i=0+jm;i<n;i++){
   int lark = 0;
   int chk = 0;
   for(lark=0;lark<point2; lark++){
      if(first[i]==calc_first[point1][lark])
      {
         chk = 1;
         break;
      }
   }
   if(chk==0)
   {
      cout<<first[i]<<", ";
      calc_first[point1][point2++] = first[i];
   }
}
cout<<"}"<<endl;
```

```
   jm=n;
   point1++;
}
cout<<"\n----------------------------------------------------------\n\n"<<endl;

char donee[cnt];
	ptr = -1;

	// Initializing the calc_follow array
	for(k = 0; k < cnt; k++) {
		for(kay = 0; kay < 100; kay++) {
			calc_follow[k][kay] = '!';
		}
	}
	point1 = 0;
	int land = 0;
	for(e = 0; e < cnt; e++)
	{
		ck = production[e][0];
		point2 = 0;
		xxx = 0;

		// Checking if Follow of ck has already been calculated
		for(kay = 0; kay <= ptr; kay++)
			if(ck == donee[kay])
				xxx = 1;

		if (xxx == 1)
			continue;
		land += 1;

		// Function call
		findfollow(ck);
		ptr += 1;
		// Adding ck to the calculated list
		donee[ptr] = ck;
		printf(" Follow(%c) = { ", ck);
		calc_follow[point1][point2++] = ck;
		// Printing the Follow Sets of the grammar
		for(i = 0 + km; i < m; i++) {
```

```
                        int lark = 0, chk = 0;
                        for(lark = 0; lark < point2; lark++)
                        {
                                if (follow[i] == calc_follow[point1][lark])
                                {
                                        chk = 1;
                                        break;
                                }
                        }
                        if(chk == 0)
                        {
                                printf("%c, ", follow[i]);
                                calc_follow[point1][point2++] = follow[i];
                        }
                }
                printf(" }\n\n");
                km = m;
                point1++;
        }
}

void findfollow(char c)
{
        int i, j;

        // Adding "$" to the follow set of the start symbol
        if(production[0][0] == c) {
                follow[m++] = '$';
        }
        for(i = 0; i < 10; i++)
        {
                for(j = 2;j < 10; j++)
                {
                        if(production[i][j] == c)
                        {
                                if(production[i][j+1] != '\0')
                                {
                                        // Calculate the first of the next Non-Terminal in the
production
                                        followfirst(production[i][j+1], i, (j+2));
```

```
                                        }

                                        if(production[i][j+1]=='\0' && c!=production[i][0])
                                        {
                                                // Calculate the follow of the Non-Terminal in the L.H.S. of
the production
                                                findfollow(production[i][0]);
                                        }
                                }
                        }
                }

}

void followfirst(char c, int c1, int c2)
{
        int k;
        // The case where we encounter a Terminal
        if(!(isupper(c)))
                follow[m++] = c;
        else
        {
                int i = 0, j = 1;
                for(i = 0; i < cnt; i++)
                {
                        if(calc_first[i][0] == c)
                                break;
                }
                //Including the First set of the Non-Terminal in the Follow of the original query
                while(calc_first[i][j] != '!')
                {
                        if(calc_first[i][j] != '#')
                        {
                                follow[m++] = calc_first[i][j];
                        }
                        else
                        {
                                if(production[c1][c2] == '\0')
                                {
                                        // Case where we reach the end of a production
```

```
                                        findfollow(production[c1][0]);
                                }
                                else
                                {
                                        // Recursion to the next symbol in case we encounter a "#"
                                        followfirst(production[c1][c2], c1, c2+1);
                                }
                        }
                        j++;
                }
        }
}
```