```
        while ( temp[i] != ']')
        {
            str[j++] = temp[i++];
        }
        str[j] = '\0';
        printf( " %d \t\t %s \t\t index\n", l, str);
        i++;
    }
    else
    {
        for ( k=0; k<8; k++)
        {
            if ( !strcmp( c[k], str))
                break;
        }
        if (k==8)
            printf( " %d \t\t %s \t\t identifier\n", l, str);
        else
            printf( " %d \t\t %s \t\t keyword\n", l, str);
    }
}
else if ( isdigit (temp[i]))
{
    j=0;
    int is_float =0, is_exp=0;
    while ( isdigit( temp[i]))
    {
        str[j++] = temp[i++];
    }
    if ( temp[i] == '.')
    {
        is_float = 1;
        str [j++] = temp[i++];
        while( isdigit (temp[i]))
        {
            str [j++] = temp[i++];
        }
    }
```

```
        i+=2;
        printf("%d \t\t*/\t\t end of comment\t\n",l);
    }
    else if((temp[i]=='/') && (temp[i+1]=='/'))
    {
        printf("%d\t\t //\t\t single line comment \t\n",l);
        break;
    }
    else if( temp[i]== " ")
    {
        j=0;
        i++;
        printf("%d\t\t\"\t\t start of literal\t\n",l);
        while( temp[i]!='"' && temp[i]!='\n')
        {
            str[j++]=temp[i++];
        }
        str[j]='\0';
        printf("%d\t\t%s\t\t end of string literal \n", l, str);
        i++;
    }
    else
    {
        j=0;
        str[j++] = temp[i];
        switch (temp[i])
        {
            case '+':
            case '-':
            case '*':
            case '/':
            case '=':
            case '>':
            case '<':
            case '%': { if ((temp[i]=='>' || temp[i]=='<' || temp[i]=='=') &&
                            temp[i+1]=='=')
```

```
        str[j++]= temp[i++];
      }
      sta[j]='\0';
      if( temp[i] == '(')
      {
         printf( "%d \t\t %s \t\t array\n",l, sta);
         i++;
         j=0
         while( temp[i] != ']')
         {
            sta[j++] = temp[i++];
         }
         sta[j]= '\0';
         printf( "%d \t\t %s \t\t index\n",l, sta);
         i++;
      }
      else
      {
         for (k = 0; k<38; k++)
         {
            if(!strcmp(c[k],sta))
            break;
         }
         if( k ==38)
            printf( "%d \t\t %s \t\t identifier\n", l, sta);
         else
            printf( "%d \t\t %s \t\t keyword \n", l, sta);
      }
   }
   else if( isdigit (temp[i]))
   {
      j=0;
      int is_float=0, is_exp=0;
      while ( isdigit( temp[i]))
      {
         sta[j++] = temp[i++];
      }
```

```
{
    i++;
    if( temp[i]=='\n')
    {
        fgets( temp, 100, fp);
        l++;
        i=0;
    }
}
i+=2
printf( "%d \t\t*/\t\t end of comment\t\n", l);
}
else if ((temp[i]=='/') && (temp[i+1]== '/'))
{
    printf( "%d \t\t // \t\t single line comment\t\n", l);
    break;
}
else if( temp[i]=='"')
{
    j=0;
    i++;
    printf("%d \t\t \" \t\t start of string literal\t\n", l);
    while( temp[i]!= '"' && temp[i]!= '\n')
    {
        str[j++] = temp[i++];
    }
    str[j]='\0';
    printf( "%d \t\t %s\"\t\t end of string literal\n ", l, str);
    i++;
}
else
{
    j=0;
    str[j++] = temp[i];
    switch( temp[i])
    {
```

Title :

Date :

Page No : 12

```
    }
  }
  fclose(fp);
}
```

input.txt

```
int main()
{
  /*
   hi welcome
  */
  // compiler design
  printf("CD");
  int a, b;
  float c = 10.6E+7;
}
```

Output:

Line No	Token	Lexeme
1	int	Keyword
1	main	Keyword
1	(	parenthesis
1	(	parenthesis
2	{	parenthesis
3	/*	start of comment
5	*/	end of comment
6	//	single line of comment
7	printf	keyword
7	(	parenthesis
7	"	start of string literal
7	CD"	end of string literal
7	)	parenthesis
7	;	punct symbol
8	int	keyword
8	a	identifier
8	,	special symbol
8	b	identifier
8	;	punct symbol
9	float	identifier
9	c	identifier
9	=	operator
9	10.6E+7	exponential
9	;	punct symbol
10	}	parenthesis

```
"="      { printf("lineno=%d \t %s \t\t assignment oper \n", lineno, yytext); }
"+"|"-"|"*"|"/"  { printf("lineno = %d \t %s \t\t operator \n", lineno,
                    yytext); }
"["|"]"|"{"|"}"|"("|")"  {printf("lineno=%d \t\t %s \t parenthesis \n",
                    lineno, yytext); }
{kw}   { printf("lineno=%d \t\t %s \t\t keyword \n", lineno, yytext); }
{arr}  { printf("lineno = %d \t %s \t\t array \n", lineno, yytext); }
{str}  { printf("lineno = %d \t %s \t\t string literal \n", lineno, yytext); }
{id}   { printf("lineno=%d \t %s \t\t identifier \n", lineno, yytext); }
{num}  { printf("lineno=%d \t %s \t\t number \n", lineno, yytext); }

%%
main(int argc, char **argv)
{
    if(argc > 1)
        yyin = fopen(argv[1], "r");
    else
        yyin = stdin;

    yylex();
}
yywrap()
{
    exit(0);
}

input.txt
int main()
{
    /*
    hi welcome
    */
    // compiler Design
    printf("CD");
```

Title : Recursive Decent Parser

Date : 5/9/24

Page No : 13

Develop a Recursive Decent Parser for statements in C.

```
#include <stdio.h>
#include <conio.h>
void E();
void E1();
void T();
void T1();
void F();
void match(char);
int flag=1;
char l,t;
main()
{
  printf("Enter input string\n");
  scanf("%c",&l);
  E();
}
void match(char t)
{
   if(l==t)
   scanf("%c",&l);
   else
   {
     flag=0;
   }
}
void E()
{
  T();
  E1();
  if( (l=='$') && (flag!=0))
  printf("successful\n");
  else
  printf("unsuccessful\n");
}
```

Grammar:

$E \longrightarrow TE'$

$E' \longrightarrow +TE' / \varepsilon$

$T \longrightarrow FT'$

$T' \longrightarrow *FT' / \varepsilon$

$F \longrightarrow i$

$F \longrightarrow (E)$

	First	Follow
$E \longrightarrow TE'$	{ i, (}	{ $,) }
$E' \longrightarrow +TE'/\varepsilon$	{ +, ε }	{ $,) }
$T \longrightarrow FT'$	{ i, (}	{ +, $,) }
$T' \longrightarrow *FT'/\varepsilon$	{ *, ε }	{ +, $,) }
$F \longrightarrow i$	{ i }	{ *, +, $,) }
$F \longrightarrow (E)$	{ (}	{ +, *, $,) }

	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to i$			$F \to (E)$		