Intermediate code generation

Generation of Quadruples for the expressions generated by the grammar

E→E+T +T
T→T*F | F
F→(E) |VAR

- **Description**

The quadruples are intermediate code forms of expressions having the following form

Operator	Operand1	Operand2	Result

One can store the quadruples in a structure as follows:

```
struct quad
{
        char op[5];
        char arg1[10];
        char arg2[10];
        char result[10];
}
```

The following program generates the quadruples for the expressions that are generated by the grammar given above. The quadruple is generated only for the productions involving operators such as E→E+T and T→ T*F. While generating the quadruples, the result will be stored in the operator1 for further use.

- **Code in quad.l**

```
%{
    #include<stdio.h>
    #include "y.tab.h"
    #include<string.h>
%}

%%

[a-z]([a-z]|[0-9])*  {  strcpy(yylval.exp,yytext);
                        return VAR;
                     }

\t    ;
\n   return 0;
.   return yytext[0];
%%
```

- **Code in quad.y**

```
%{
 #include<stdio.h>
#include<string.h>
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];

int i=0,j;
%}

%union
{
char exp[10];
}

%token <exp> VAR
%type <exp> S E T F

%%
S: E   {  printf("\n THere are %d quadrupls n",i);
          printf("\n List of Quadruples are: \n");
             for(j=0;j<i;j++)

                printf("%s\t%s\t%s\t%s\n",QUAD[j].op,QUAD[j].arg1,
          QUAD[j].arg2,QUAD[j].result);
        }
   ;
E: E'+'T { printf("\n E ->E+T,  $1=%s,  $3=%s,$$=%s\n",$1,$3,$$);
           strcpy(QUAD[i].op,"+");
           strcpy(QUAD[i].arg1,$1);
           strcpy(QUAD[i].arg2,$3);
           strcpy(QUAD[i].result,$$);i++;
           i++;
         }
 | T { printf("\n E -> T,   $1=%s,   $$=%s\n",$1,$$);}
 ;
T: T'*'F { printf("\n T -> T*F,  $1=%s,  $3=%s,
                          $$=%s\n",$1,$3,$$);
          strcpy(QUAD[i].op,"*");
          strcpy(QUAD[i].arg1,$1);
          strcpy(QUAD[i].arg2,$3);
          strcpy(QUAD[i].result,$$);
          i++;
```

```
          }
 | F { printf("\n T -> F,  $1=%s,  $$=%s\n",$1,$$);}
 ;
F:  VAR {printf("\n F ->VAR and $1=%s,   $$=%s \n",$1,$$);}
 ;

%%
main()
{
  yyparse();
}

int yywrap(){
    return 1;
}
void yyerror(char *s)
{
  printf("%s",s);
}
```

Execute the above code for sample expressions below and observe the list of quadruples generated. Print statements are included for understanding the order in which the quadruples are being generated. You can comment the print statements.

1. a+b*c
2. a+c+d*f+e
3. a+*b