**Name: Niranjan Vinod Patil.**

**Batch: B31.**

**Roll No: SCETTYB305.**

**Course: Compiler Design.**

**----------------------------------------------------------------------------------------------------**

**Assignment No: 06**

**Aim:**

Write an attributed translation grammar to recognize declaration of simple variables, control structure as per syntax of c++.

**Program**

icg.l  
  
%{  
   #include "icg.tab.h"  
   extern char yyval;  
%}  
  
NUMBER [0-9]+  
LETTER [a-zA-Z]+  
  
%%  
{NUMBER} {yylval.sym=(char)yytext[0]; return NUMBER;}  
{LETTER} {yylval.sym=(char)yytext[0];return LETTER;}  
  
\n {return 0;}  
. {return yytext[0];}  
  
%%

icg.y

%{  
  
  #include<stdio.h>  
  #include<string.h>  
  #include<stdlib.h>  
void ThreeAddressCode();  
void triple();  
void qudraple();  
char AddToTable(char ,char, char);  
  
  int ind=0;  
  char temp='A';  
  struct incod  
  {  
    char opd1;  
    char opd2;  
    char opr;  
  };  
%}  
  
%union  
{  
 char sym;  
}  
  
%token <sym> LETTER NUMBER  
%type <sym> expr  
%left '-''+'  
%left '\*''/'  
  
%%  
  
statement: LETTER '=' expr ';' {AddToTable((char)$1,(char)$3,'=');}  
           | expr ';'  
       ;  
  
expr: expr '+' expr {$$ = AddToTable((char)$1,(char)$3,'+');}  
      | expr '-' expr {$$ = AddToTable((char)$1,(char)$3,'-');}  
      | expr '\*' expr {$$ = AddToTable((char)$1,(char)$3,'\*');}  
      | expr '/' expr {$$ = AddToTable((char)$1,(char)$3,'/');}  
      | '(' expr ')' {$$ = (char)$2;}  
      | NUMBER {$$ = (char)$1;}  
      | LETTER {$$ = (char)$1;}  
      ;  
  
%%  
  
yyerror(char \*s)  
{  
  printf("%s",s);  
  exit(0);  
}  
  
struct incod code[20];  
  
int id=0;  
  
char AddToTable(char opd1,char opd2,char opr)  
{  
code[ind].opd1=opd1;  
code[ind].opd2=opd2;  
code[ind].opr=opr;  
ind++;  
temp++;  
return temp;  
}  
  
void ThreeAddressCode()  
{  
int cnt=0;  
temp++;  
printf("\n\n\t THREE ADDRESS CODE\n\n");  
while(cnt<ind)  
{  
    if(code[cnt].opr!='=')  
    {  
    printf("%c : = \t",temp);  
    }  
      
        if(isalpha(code[cnt].opd1))  
        printf("%c\t",code[cnt].opd1);  
    else  
        {printf("%c\t",temp);}  
  
    printf("%c\t",code[cnt].opr);  
      
    if(isalpha(code[cnt].opd2))  
        printf("%c\t",code[cnt].opd2);  
    else  
        {printf("%c\t",temp);}  
  
    printf("\n");  
    cnt++;  
    temp++;  
}  
}  
  
void quadraple()  
{  
    int cnt=0;  
temp++;  
printf("\n\n\t QUADRAPLE CODE\n\n");  
while(cnt<ind)  
{  
    //printf("%c : = \t",temp);  
  
      
          printf("%d",id);  
          printf("\t");          
          printf("%c",code[cnt].opr);  
          printf("\t");       
          
        if(isalpha(code[cnt].opd1))  
                printf("%c\t",code[cnt].opd1);  
        else   
        printf("%c\t",temp);  
  
            //printf("%c\t",code[cnt].opr);  
      
        if(isalpha(code[cnt].opd2))  
            printf("%c\t",code[cnt].opd2);  
        else   
        { printf("%c\t",temp);}  
          
        if(code[cnt].opr!='=')  
        printf("%c",temp);  
  
    printf("\n");  
    cnt++;  
    temp++;  
    id++;  
      
}  
}  
  
void triple()  
{  
    int cnt=0,cnt1,id1=0;  
temp++;  
printf("\n\n\t TRIPLE CODE\n\n");  
while(cnt<ind)  
{  
    //printf("%c : = \t",temp);  
  
        if(id1==0)  
        {  
            printf("%d",id1);  
          printf("\t");          
          printf("%c",code[cnt].opr);  
          printf("\t");       
           if(isalpha(code[cnt].opd1))  
        printf("%c\t",code[cnt].opd1);  
            else  
        {printf("%c\t",temp);}  
  
            //printf("%c\t",code[cnt].opr);  
        cnt1=cnt-1;  
        if(isalpha(code[cnt].opd2))  
            printf("%c",code[cnt].opd2);  
        else  
        {printf("%c\t",temp);}  
        }  
        else  
        {  
          printf("%d",id1);  
          printf("\t");          
          printf("%c",code[cnt].opr);  
          printf("\t");       
           if(isalpha(code[cnt].opd1))  
        printf("%c\t",code[cnt].opd1);  
            else  
        {printf("%c\t",temp);}  
  
            //printf("%c\t",code[cnt].opr);  
        cnt1=cnt-1;  
        if(isalpha(code[cnt].opd2))  
            printf("%d",id1-1);  
        else  
        {printf("%c\t",temp);}  
        }  
          
  
    printf("\n");  
    cnt++;  
    temp++;  
    id1++;  
      
}  
  
}  
  
main()  
{  
 printf("\nEnter the Expression: ");  
 yyparse();  
temp='A';  
ThreeAddressCode();  
temp='A';  
quadraple();  
temp='A';  
triple();  
}  
  
yywrap()  
{  
 return 1;  
}

**Output**

//........................output..........................................  
/\*Steps  
[student@student ~]$ lex th.l  
[student@student ~]$ yacc -d th.y  
[student@student ~]$ gcc lex.yy.c y.tab.c  
[student@student ~]$ lex th.l  
  
  
  
syntax error[student@student ~]$ ./a.out  
  
Enter the Expression: a=10+20;  
  
  
     THREE ADDRESS CODE  
  
B : =     B    +    B      
C : =     a    =    B      
  
  
     QUADRAPLE CODE  
  
0    +    E    E    E  
1    =    a    B    F  
  
  
     TRIPLE CODE  
  
0    +    H    H      
1    =    a    0  
  
  
  
  
  
  
output2:  
Enter the Expression: a=20+30-10;  
  
  
     THREE ADDRESS CODE  
  
B : =     B    +    B      
C : =     B    -    C      
D : =     a    =    C      
  
  
     QUADRAPLE CODE  
  
0    +    F    F    F  
1    -    B    G    G  
2    =    a    C    H  
  
  
     TRIPLE CODE  
  
0    +    J    J      
1    -    B    K      
2    =    a    1  
\*/