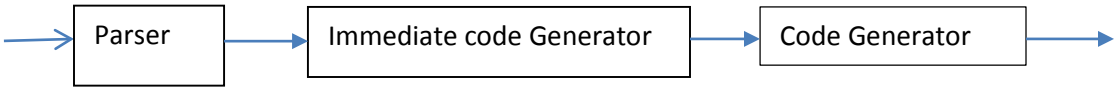
USES OF THREE ADDRESS CODE

Since three-address code is used as an intermediate language within compilers, the operands will not be memory addresses or processor registers, but rather (temporary) symbolic addresses that will be translated into actual addresses during register allocation. It is also not uncommon that operand names are numbered sequentially since three-address code is typically generated by the compiler.



It is an abstract form of immediate code. In a compiler, these statements can be implemented as records with fields for the operator and operands. Three such representations are Quadruples, Triples and Indirect triples.

Example:-

Expression: a=b+c*d--d;w=x+y

t0 = b

t1 = c

t2 = d

t3 = t1 * t2

t4 = t0 + t3

t5 = d

t6 = -t5

t7 = t4 - t6

a = t7

t8 = x

t9 = y

t10 = t8 + t9

w = t10

Objective

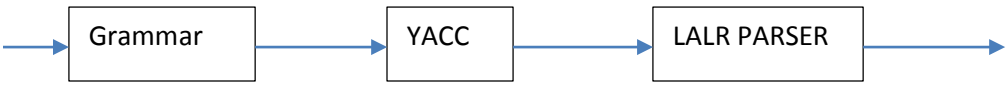
Code generation is the process by which a compiler's code generator converts some intermediate representation of source code into a form (e.g., machine code) that can be readily executed by a machine.

So we are aiming to create a program that converts a "C Code" to a "Three Address code"

Our target is to achieve the Three address code using:-

1) Lexical Analyzer -: its main job is to break up an input stream into more usable elements or in, other words, to identify the "interesting bits" in a text file. It is use to generate tokens for the input file.

2) Syntax Directed Translation -: the translation of languages guided by context free grammars. Values of the attributes are computed by "semantic rules" associated with grammar productions.



Yacc Stands for "YET ANOTHER COMPILER COMPILER".

It can be applied to almost any situation where text-based input is being used. The parser also verifies that the input is syntactically sound.

GRAMMER USED (for, while, do-while, regular expression)

```
Z       :       '{' Y
        ;

Y       :       A Y
        |       B Y
        |       C Y
        |       N Y
        |       '}'
        ;

A       :       FOR ‘(‘ S ‘;’ K ‘;’ S ‘)’’{‘ N ‘}’
        ;

B       :       WHILE '('  K ')' '{' N '}'
        ;

C       :       DO  '{' N '}'
                WHILE '('  K ')' ';'
        ;

K       :       E OPR1 E
        |       E OPR2 E
        ;

N       :       S ';' N
        |       S
        ;

S       :       ID '=' E
         ;

E       :       E '+'  T
        |       E '-' T
        |        T
        ;

T       :       T '*' F
        |       T '/' F
        |       F
         ;

F        :      '(' E ')'
        |       '-' F
        |       ID
        |       NUM  ;
```

LEX CODE FOR TOKENS

```
[ \t\n]+            {}

"do"                {return(DO);}

"while"             { return(WHILE);}

"for"               { return(FOR);}

{ALPHA}({ALPHA}|{DIGIT})*     { yylval.str=strdup(yytext);return(ID);}

{DIGIT}+        {yylval.str=strdup(yytext);
                 return(NUM);
                 }

"<="|">="|"<"|">"       {yylval.str=strdup(yytext);
                         return (OPR1);
                         }

"=="|"!="       {yylval.str=strdup(yytext);
                 return(OPR2);
                 }

.           {  return yytext[0];}
```