COMPILER DESIGN

ASSIGNMENT EVALUATION

Date:4/4/2018

GROUP 4

Athira S Pillai (15)
Arundhati Kurup(14)
Athul Raj (16)
Arjun C Ramesh(13)

Evaluation of :

1.Arithmetic expressions with nested parenthesis

Sample input : ((1+2)*((4*5)+3))

2.Trigonometric expressions

Sample input : sin(30)+cos(0)

3.Mathematical expressions

Sample input : sqrt(5)/sqrt(2)

4.Logical expressions

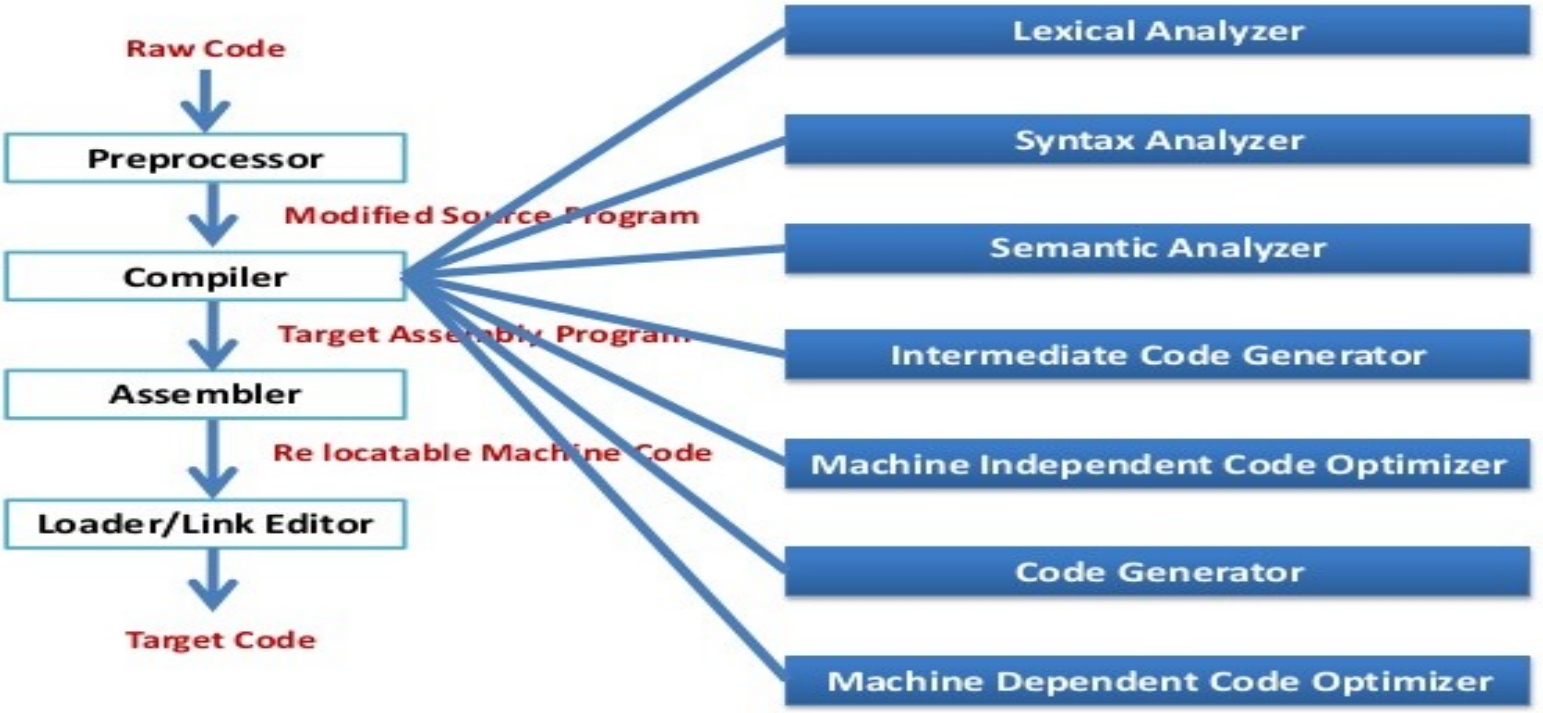
Sample input : 1&1 , !1 , (1|1)&(!0)

*combinations

sin(30)+12-5*cos(90)

DESIGNED PHASES

1.Lexical analysis
2.Syntax analysis
3.Semantic analysis
4.Intermediate code generation (Three address code)



Challenges and Scope for improvement

- **Three address including trigonometric expressions**
- **Variable assignments using symbol table and further evaluations**
- **Semantic checking for complex logical expressions and including relational expressions**
- **Including more math functions like power , roots (except square root) , ncr,npr etc**

PHASE 1 (using lex tool)

```
DIGIT [0-9]+\.?|[0-9]*\.[0-9]+

{DIGIT}    {yylval=atof(yytext);return NUM;}
"exit"          {return exit_command;}
cos|COS       {return COS;}
sin|SIN   {return SIN;}
tan|TAN  {return TAN;}
log|LOG {return LOG;}
sqrt|SQRT     {return SQRT;}


\n|.          {return yytext[0];}
```

PHASE 2 (using yacc tool)

```
S       : S E '\n' { printf("Answer: %g \nEnter next expression:\n", $2); }
        | S '\n'
        |
        | error '\n' { yyerror("Error: Enter once more...\n" );yyerrok; }

        ;
E       : E '+' E    { $$ = $1 + $3; }
        | E'-'E    { $$=$1-$3; }
        | E'*'E    { $$=$1*$3; }

         |E'/'E    { $$=$1/$3; }
         |E'&'E              { $$=(int)$1&(int)$3; }
         |E'|'E        { $$=(int)$1|(int)$3; }


        | '('E')'    { $$=$2; }

        | NUM
        | COS'('E')' {$$=cos($3);}
        | SIN'('E')' {$$=sin($3);}
        | TAN'('E')' {$$=tan($3);}
        | LOG'('E')' {$$=log($3);}
      | SQRT'('E')' {$$=sqrt($3);}
      | exit_command          {exit(EXIT_SUCCESS);}
```

PHASE 3 (3 address generation)

```
prog : prog expr '\n' {printf("%c = %c\n",p,p-1);}
|
;
expr : STR
| expr '+' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'+',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'+',$3);p++;}}
| expr '-' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'-',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'-',$3);p++;}}
| expr '*' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'*',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'*',$3);p++;}}
| expr '/' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'/',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'/',$3);p++;}}
| expr '%' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'%',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'%',$3);p++;}}
| expr '&' expr {if(i==0){ printf("%c = %d %c %d\n",p,$1,'&',$3); p++;i++;}
                 else{ printf("%c = %c %c %d\n",p,p-1,'&',$3);p++;}}
```