Nabla

Lexer Design

Team 9

Quick Intro to Nabla

Simple

Domain specific

Statically typed programming language

Developed for simplifying tensors computation

Supports automatic-differentiation

Implements a dynamic computational graph in the back-end.

Dive into Lexer

Taken reference from C Grammar

We use our own keywords for various instructions

Using Flex for our *lexer.l* file to generate symbol table and mapping

Operators and in-built functions supported

cos, sin, log, exp, grad, backward
*+, -, *, /, >>, <<, &, |, ^, %*
@ (matrix multiplication operator)

Code Snippet

```
"@"         { count(); return AT_OP; }
">>="           { count(); return RIGHT_ASSIGN; }
"<<="           { count(); return LEFT_ASSIGN; }
"+="            { count(); return ADD_ASSIGN; }
"-="            { count(); return SUB_ASSIGN; }
"*="            { count(); return MUL_ASSIGN; }
"/="            { count(); return DIV_ASSIGN; }
"%="            { count(); return MOD_ASSIGN; }
"&="            { count(); return AND_ASSIGN; }
"^="            { count(); return XOR_ASSIGN; }
"|="            { count(); return OR_ASSIGN; }
"@="            { count(); return AT_ASSIGN;}
">>"            { count(); return RIGHT_OP; }
"<<"            { count(); return LEFT_OP; }
"++"            { count(); return INC_OP; }
"--"            { count(); return DEC_OP; }
"&&"            { count(); return AND_OP; }
"||"            { count(); return OR_OP; }
"<="            { count(); return LE_OP; }
">="            { count(); return GE_OP; }
"=="            { count(); return EQ_OP; }
"!="            { count(); return NE_OP; }
```

```
";"         { count(); return ';'; }
"{"         { count(); return '{'; }
"}"         { count(); return '}'; }
","         { count(); return ','; }
":"         { count(); return ':'; }
"="         { count(); return '='; }
"("         { count(); return '('; }
")"         { count(); return ')'; }
"["         { count(); return '['; }
"]"         { count(); return ']'; }
"."         { count(); return '.'; }
"&"         { count(); return '&'; }
"!"         { count(); return '!'; }
"~"         { count(); return '~'; }
"-"         { count(); return '-'; }
"+"         { count(); return '+'; }
"*"         { count(); return '*'; }
"/"         { count(); return '/'; }
"%"         { count(); return '%'; }
"<"         { count(); return '<'; }
">"         { count(); return '>'; }
"^"         { count(); return '^'; }
"|"         { count(); return '|'; }
"?"         { count(); return '?'; }
```

```
"char"          { count(); return CHAR; }
"cns"           { count(); return CNS; }
"elif"          { count(); return ELIF; }
"else"          { count(); return ELSE; }
"endif"         {count(); return ENDIF;}
"float"         { count(); return FLOAT; }
"loop"          { count(); return LOOP; }
"if"            { count(); return IF; }
"int"           { count(); return INT; }
"Tensor"        { count(); return TENSOR; }
"var"           { count(); return VAR; }
"bool"          { count(); return BOOL;}

"sizeof"        { count(); return SIZEOF;}
"grad"          { count(); return GRAD;}
"backward"      { count(); return BACKWARD;}
"cos"           { count(); return COS;}
"sin"           { count(); return SIN;}
"exp"           {count(); return EXP;}
"log"           {count(); return LOG;}
"print"         {count(); return PRINT;}
```

Sample Code

```
{
    var Tensor a[2][2]=[[1,2],[3,4]];  //declaring tensor a
    bool ans;
    var Tensor b[10][2][2];
    loop(int i=1;i<10;i++)
    {   b[0][2][2]=a[2][2];
        if(i-1>=0)
    {
        b[i][2][2]=b[i-1][2][2]@a;
        if(b[i][2][2]==a[2][2])
        {
           ans=True;
        }
        else
        {
           ans=False;
        }
        endif
     }
    }
      print("Is this idempotent matrix?");
      print(ans);
}
```

Sample Output

```
{
VAR TENSOR IDENTIFIER [ CONSTANT ] [ CONSTANT ] = [ [ CONSTANT , CONSTANT ] , [ CONSTANT , CONSTANT ] ] ;
BOOL IDENTIFIER ;
VAR TENSOR IDENTIFIER [ CONSTANT ] [ CONSTANT ] [ CONSTANT ] ;
LOOP ( INT IDENTIFIER = CONSTANT ; IDENTIFIER < CONSTANT ; IDENTIFIER INC_OP )
{ IDENTIFIER [ CONSTANT ] [ CONSTANT ] [ CONSTANT ] = IDENTIFIER [ CONSTANT ] [ CONSTANT ] ;
IF ( IDENTIFIER - CONSTANT GE_OP CONSTANT )
{
IDENTIFIER [ IDENTIFIER ] [ CONSTANT ] [ CONSTANT ] = IDENTIFIER [ IDENTIFIER - CONSTANT ] [ CONSTANT ] [ CONSTANT ] AT_OP IDENTIFIER ;
IF ( IDENTIFIER [ IDENTIFIER ] [ CONSTANT ] [ CONSTANT ] EQ_OP IDENTIFIER [ CONSTANT ] [ CONSTANT ] )
{
IDENTIFIER = IDENTIFIER ;
}
ELSE
{
IDENTIFIER = IDENTIFIER ;
}
ENDIF
}
}
PRINT ( STRING_LITERAL ) ;
PRINT ( IDENTIFIER ) ;
}
```