Including only changes made in the source code, everything else remains the same;

calculator_partial.l

Lexmes:

```
">="    return GE;
"<="    return LE;
"=="    return EQ;
"!="    return NE;
"while" return WHILE;
"if"    return IF;
"else"  return ELSE;
```

Regular expressions:

```
[=+*;/(){}.%-]       { return *yytext; }

[a-zA-Z][a-zA-Z0-9]*       {//printf("user defined variable: %s\n", yytext);
                                yylval.symbol_index = hash(yytext);
                                //printf("position: %d\n", hash(yytext));
                                return VARIABLE; }
```

The definition defines user defined inputs, which can start with lower/capital case letters and followed by letters and digits.
User defined variables cannot start with digits.
example: it can take input1, number10, result, X1, Y1, etc. 10input will be an error.

Hash function:

Hashing the user defined variables into symbol table; taking unspecified number of variables.

```
int hash (char *input)
{
   unsigned int hash = 0;
   for (int i = 0 ; input[i] != '\0' ; i++)
   {
      hash = (31*hash + input[i])%1000;
   }

   return hash;
}
```

calculator_partial.y

Tokens:

```
%token <input_Value> INTEGER
%token <symbol_index> VARIABLE
%token PRT
%token WHILE IF
%nonassoc ELSE
%nonassoc IFX

%left '+'
%left '-'
%left '*'
%left '/'
%left '%'
%right '='
%nonassoc UMINUS
%left GE NE LE EQ '<' '>'
```

BNF:

```
%type <nodePointer> stmt expr term factor stmt_list

stmt:
    ';'                                        { $$ = opera(';', 2, NULL, NULL); }
    | expr ';'                                 { $$ = $1; }
    | PRT expr ';'                             { $$ = opera(PRT, 1, $2); }
    | VARIABLE '=' expr ';'                    { $$ = opera('=', 2, identifier($1), $3); }
    | WHILE '(' expr ')' stmt                  { $$ = opera(WHILE, 2, $3, $5); }
    | IF '(' expr ')' stmt %prec IFX           { $$ = opera(IF, 2, $3, $5); }
    | IF '(' expr ')' stmt ELSE stmt           { $$ = opera(IF, 3, $3, $5, $7); }
    | '{' stmt_list '}'                        { $$ = $2; }
    ;

stmt_list:
    stmt                                       { $$ = $1; }
    |stmt_list stmt                            { $$ = opera(';', 2, $1, $2); }
    ;

expr:
     expr '+' term                             { $$ = opera('+', 2, $1, $3); }
    | expr '-' term                            { $$ = opera('-', 2, $1, $3); }
    | term                                     { $$ = $1; }
    | expr '<' expr                            { $$ = opera('<', 2, $1, $3); }
    | expr '>' expr                            { $$ = opera('>', 2, $1, $3); }
    | expr GE expr                             { $$ = opera(GE, 2, $1, $3); }
```

```
  | expr LE expr                    { $$ = opera(LE, 2, $1, $3); }
  | expr NE expr                    { $$ = opera(NE, 2, $1, $3); }
  | expr EQ expr                    { $$ = opera(EQ, 2, $1, $3); }
  ;
```

calculator_interpreter_partial.c

```
case WHILE:
        while(interpret(p->operator_.poperands[0]))
            interpret(p->operator_.poperands[1]);
        return 0;
case IF:
        if( interpret(p->operator_.poperands[0]))
            interpret(p->operator_.poperands[1]);
        else if( ! interpret(p->operator_.poperands[0]))
            interpret(p->operator_.poperands[2]);
        return 0;
case '<':
        return interpret(p->operator_.poperands[0]) < interpret(p->operator_.poperands[1]);
case '>':
        return interpret(p->operator_.poperands[0]) > interpret(p->operator_.poperands[1]);
case GE:
        return interpret(p->operator_.poperands[0]) >= interpret(p-
                        >operator_.poperands[1]);
case LE:
        return interpret(p->operator_.poperands[0]) <= interpret(p-
                        >operator_.poperands[1]);
case NE:
        return interpret(p->operator_.poperands[0]) != interpret(p->operator_.poperands[1]);
case EQ:
        return interpret(p->operator_.poperands[0]) == interpret(p-
                        >operator_.poperands[1]);
```

There are four source files contained in the file:

calculator_interpreter_partial.h

calculator_interpreter_partial.c

calculator_partial.l

calculator_partial.y.

The symbol table size has been changed to 1000 instead of 26 in both calculator_interpreter_partial.h and calculator_interpreter_partial.c.

The following is how I compile and use the interpreter on Mac OS terminal:

Download the given "partial folder". Substitute the four altered source files into the folder. First locate the source file folder in the terminal. There is a .sh file called build_partial.sh, run it in the terminal. It would create a executable file in the source folder, and some other necessary files if successfully complied. Then run the executable; we can test the calculator with our input there.

Demonstration: