LAN Compiler

Huizhu Zhang

December 1, 2020

Language Description

- This language is called LAN. It should be capable of simple decimal algebra operations on integers.
- The only keyword in LAN: **int**

- Example:

```
int a = 6;
int b = 10;
a + b; (output: 16)
int c = 0;
a / c; (error)
```

Lexical Tokens

- The concrete syntax of LAN is based on ASCII character encoding.

<identifier>	::=	[A-Za-z_]*
<number>	::=	0 | [1-9][0-9]*
<binop>	::=	+ | - | * | / | %
<asnop>	::=	=

Precedence (not yet implemented)

- Precedence of all operations must be specified with parenthesis | ()

- Example:

```
int a = 6;
int b = 5;
int c = 4;
1 + (a - b) / c; (output: 1)
```

Whitespace and Token Delimiting

- In LAN, whitespace is either a space or a carriage return (\r) in ASCII encoding.
- Whitespace is ignored, except that it terminates tokens. For example, a+b is one token, while a + b is three tokens.

Language Syntax

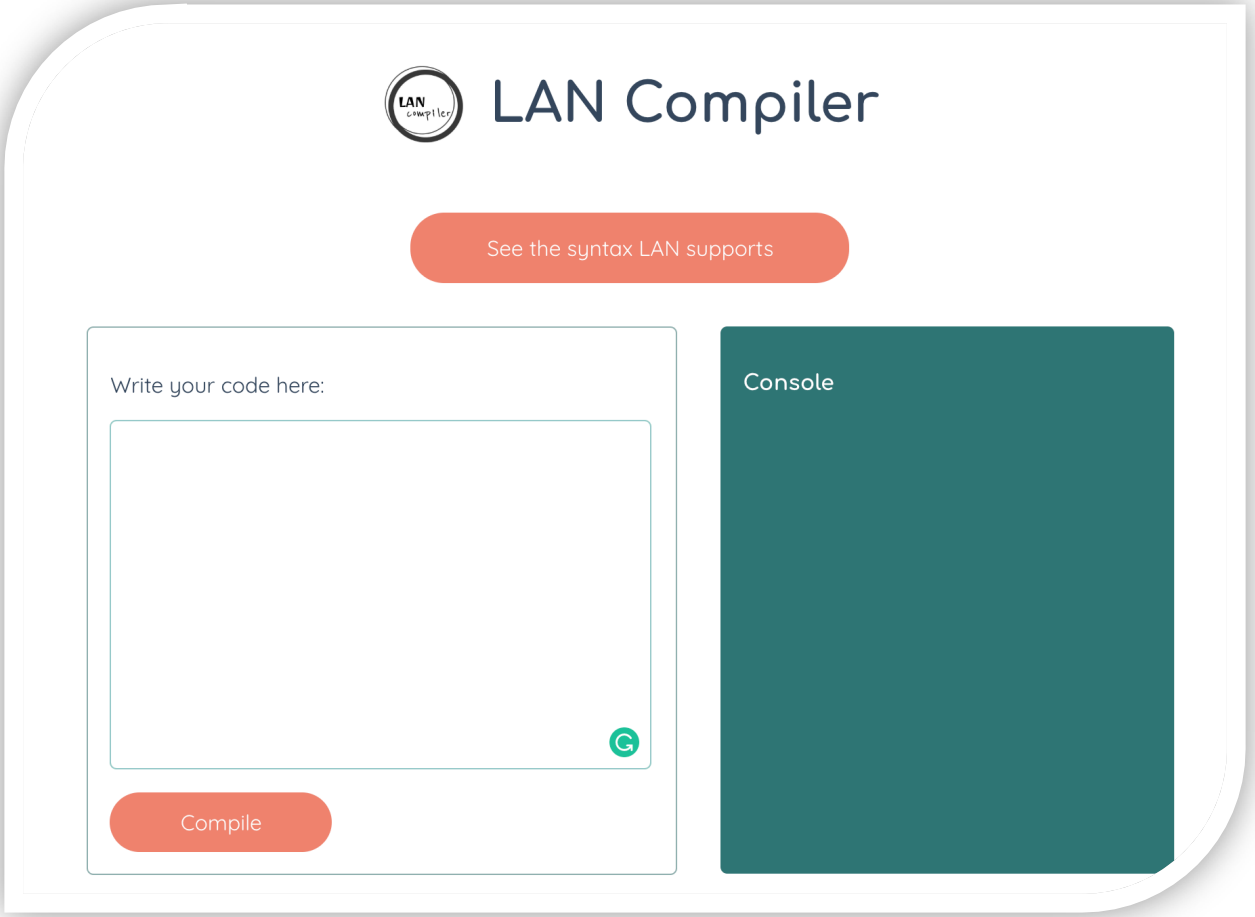
- The compiler translates source programs written in LAN. The syntax of LAN is defined by the context-free grammar shown here
- *Non-terminals are in <brackets>, terminals are in **bold**.

<statements>	::=	<statements> | <statement>
<statement>	::=	<declaration>;
<declaration>	::=	**int identifier** = <expression>;
<expression>	::=	**identifier** | **number** | <expression> <binop> <expression>

Language Semantics

- Declaration
  - Though declarations are a bit redundant in a language with only one type, LAN requires every variable to be declared (with the correct type, in this case **int**) and initialized (by given a value) before being used.
- Operational semantics
  - The division i/k returns the truncated quotient of the division of i by k, dropping any fractional part. This means it always rounds towards zero.
  - The modulus i % k returns the remainder of the division of i by k.
  - Division i/k and modulus i % k are required to raise a divide exception if k = 0.

LAN User Interface Written with React and JavaScript



Source Code

Three major functions: lexer, parser and execute

- The lexer turns the input string into an array of tokens.
  - Example
    - Input: int a = 4; int b = 3; a + b;
    - Returns: [‘int’, ‘a’, ‘=’, ‘4’, ‘int’, ‘b’, ‘=’ , ‘3’, ‘a’, ‘+’, ‘b’]

```
lexer = (str) => {
    return str.replace(/[;\n\r]/g, ' ').split(' ').filter(token => token.length).map((token) => {
        return isNaN(token) ? { type: 'word', value: token } : { type: 'number', value: token }
    })
};
```

Sourc

Three ma

- The par tax Tree)
  - Exam
    - In
    - O

```
if (current_token.value === 'int') {
    let declaration = {
        type: 'declaration',
        identifier: {
            name: '',
            value: null
        },
    }
    let identifier = tokens.shift().value;
    if (this.isValidIdentifier(identifier) && !isIdentifierDeclared(identifier)) {
        declaration.identifier.name = identifier;
        // Identifier needs to be followed by an assignment operator
        if (tokens[0] && tokens[0].value === '=') {
            tokens.shift();
            if (tokens[0] && !isNaN(tokens[0].value)) {
                declaration.identifier.value = parseInt(tokens.shift().value);
                declaredIdentifiers.push({
                    name: declaration.identifier.name,
                    value: declaration.identifier.value
                })
                AST.body.push(declaration);
            } else {
                this.props.addErrorMessage('Value should be assigned to identifier!');
            }
        } else {
            this.props.addErrorMessage('Identifier is undefined!');
        }
    } else {
        this.props.addErrorMessage('Identifier is not valid!');
    }
```

So

Thr

- T ult
  o

```
execute = (AST) => {
    AST.body.forEach(el => {
        if (el.type === 'expression') {
            let leftHand = el.expressions[0];
            let rightHand = el.expressions[1];
            let operator = el.operator;
            if ((operator === '/' || '%') && rightHand == 0) {
                this.props.addErrorMessage('Algebra 101, nothing can be divided by zero.');
            } else {
                switch (operator) {
                    case '+':
                        this.props.addOutput(parseInt(leftHand + rightHand));
                        break;
                    case '-':
                        this.props.addOutput(parseInt(leftHand - rightHand));
                        break;
                    case '*':
                        this.props.addOutput(parseInt(leftHand * rightHand));
                        break;
                    case '/':
                        this.props.addOutput(parseInt(leftHand / rightHand));
                        break;
                    case '%':
                        this.props.addOutput(parseInt(leftHand % rightHand));
                        break;
```

Demonstration

- https://huizhuz.github.io/CompilerLan/

Code Example:

```
int a = 4;
int b;
a / b;
```