Testing Results

Test 1: int

- The compiler takes in a simple additions source code piece of code and outputs the tokens as follows

Output

Test program

```
{
        int a
        a = 4

        int b
        b = 2 + a
} $
```

Tokens

```
Line 1: T_LBRACE [ { ]

Line 2: T_INT [ int ]

Line 2: T_ID [ a ]

Line 3: T_ID [ a ]

Line 3: T_SINGLE_EQUALS [ = ]

Line 3: T_DIGIT [ 4 ]

Line 5: T_INT [ int ]

Line 5: T_ID [ b ]

Line 6: T_ID [ b ]

Line 6: T_SINGLE_EQUALS [ = ]

Line 6: T_DIGIT [ 2 ]

Line 6: T_PLUS [ + ]

Line 6: T_ID [ a ]

Line 7: T_RBRACE [ } ]

Line 7: T_EOF [ $ ]
```

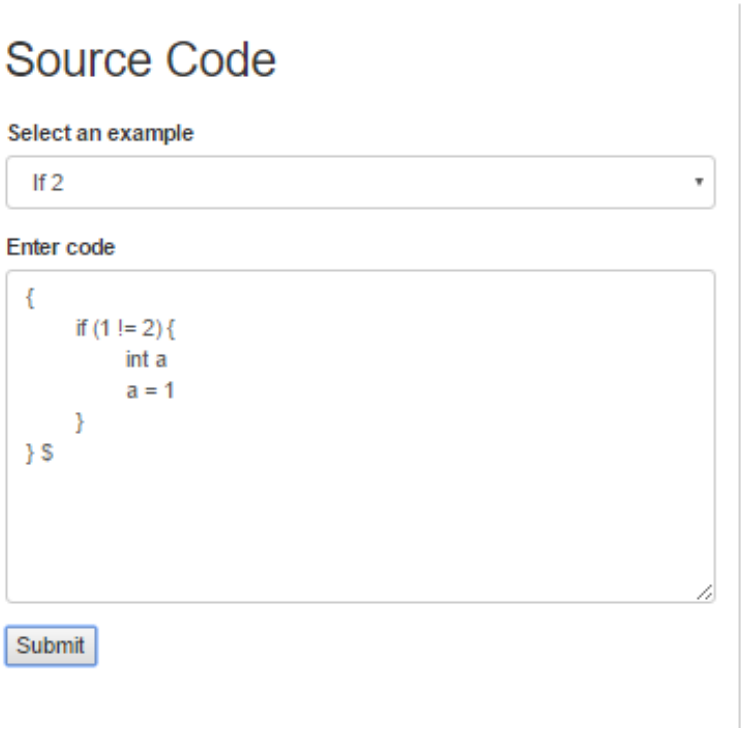
Test 2: String

- The compiler takes in a string and produces the tokens as follows

Tokens

```
Line 1: T_LBRACE [ { ]
Line 2: T_INT [ int ]
Line 2: T_ID [ a ]
Line 3: T_ID [ a ]
Line 3: T_SINGLE_EQUALS [ = ]
Line 3: T_DIGIT [ 4 ]
Line 4: T_IF [ if ]
Line 4: T_LPAREN [ ( ]
Line 4: T_ID [ a ]
Line 4: T_DOUBLE_EQUALS [ == ]
Line 4: T_DIGIT [ 4 ]
Line 4: T_RPAREN [ ) ]
Line 4: T_LBRACE [ { ]
Line 5: T_PRINT [ print ]
Line 5: T_LPAREN [ ( ]
Line 5: T_QUOTE [ " ]
Line 5: T_ID [ h ]
Line 5: T_ID [ e ]
Line 5: T_ID [ l ]
Line 5: T_ID [ l ]
Line 5: T_ID [ o ]
Line 5: T_WHITE_SPACE [   ]
Line 5: T_ID [ w ]
Line 5: T_ID [ o ]
Line 5: T_ID [ r ]
Line 5: T_ID [ l ]
Line 5: T_ID [ d ]
Line 5: T_QUOTE [ " ]
Line 5: T_RPAREN [ ) ]
Line 6: T_RBRACE [ } ]
```

Test 3: if



- The compiler takes in source code that represents an if statement and returns all the tokens within the source code

Output

Test program

```
{
	if (1 != 2) {
		int a
		a = 1
	}
} $
```

Tokens

```
Line 1: T_LBRACE [ { ]

Line 2: T_IF [ if ]

Line 2: T_LPAREN [ ( ]

Line 2: T_DIGIT [ 1 ]

Line 2: T_NOT_EQUALS [ != ]

Line 2: T_DIGIT [ 2 ]

Line 2: T_RPAREN [ ) ]

Line 2: T_LBRACE [ { ]

Line 3: T_INT [ int ]

Line 3: T_ID [ a ]

Line 4: T_ID [ a ]

Line 4: T_SINGLE_EQUALS [ = ]

Line 4: T_DIGIT [ 1 ]

Line 5: T_RBRACE [ } ]

Line 6: T_RBRACE [ } ]

Line 6: T_EOF [ $ ]
```

Test 4: While

Source Code

Select an example

While

Enter code

```
{
	int x
	x = 0

	while (x != 5)
	{
		print(x)
		x = 1 + x
	}
} $
```

Submit

Output

Test program

```
{
	int x
	x = 0

	while (x != 5)
	{
		print(x)
		x = 1 + x
	}
} $
```

Tokens

```
Line 1: T_LBRACE [ { ]

Line 2: T_INT [ int ]

Line 2: T_ID [ x ]

Line 3: T_ID [ x ]

Line 3: T_SINGLE_EQUALS [ = ]

Line 3: T_DIGIT [ 0 ]

Line 5: T_WHILE [ while ]

Line 5: T_LPAREN [ ( ]

Line 5: T_ID [ x ]

Line 5: T_NOT_EQUALS [ != ]

Line 5: T_DIGIT [ 5 ]

Line 5: T_RPAREN [ ) ]

Line 6: T_LBRACE [ { ]

Line 7: T_PRINT [ print ]

Line 7: T_LPAREN [ ( ]

Line 7: T_ID [ x ]

Line 7: T_RPAREN [ ) ]

Line 8: T_ID [ x ]

Line 8: T_SINGLE_EQUALS [ = ]

Line 8: T_DIGIT [ 1 ]

Line 8: T_PLUS [ + ]

Line 8: T_ID [ x ]

Line 9: T_RBRACE [ } ]

Line 10: T_RBRACE [ } ]

Line 10: T_EOF [ $ ]
```

Test 5: Boolean

Source Code

Select an example

Boolean

Enter code

```
{
	int a
	a = 1

	boolean b
	b = (true == (true != (false == (true != (false != (a == a))))))

	print(b)
} $
```

Submit

Tokens

```
Line 1: T_LBRACE [ { ]
Line 2: T_INT [ int ]
Line 2: T_ID [ a ]
Line 3: T_ID [ a ]
Line 3: T_SINGLE_EQUALS [ = ]
Line 3: T_DIGIT [ 1 ]
Line 5: T_BOOLEAN [ boolean ]
Line 5: T_ID [ b ]
Line 6: T_ID [ b ]
Line 6: T_SINGLE_EQUALS [ = ]
Line 6: T_LPAREN [ ( ]
Line 6: T_TRUE [ true ]
Line 6: T_DOUBLE_EQUALS [ == ]
Line 6: T_LPAREN [ ( ]
Line 6: T_TRUE [ true ]
Line 6: T_NOT_EQUALS [ != ]
Line 6: T_LPAREN [ ( ]
Line 6: T_FALSE [ false ]
Line 6: T_DOUBLE_EQUALS [ == ]
Line 6: T_LPAREN [ ( ]
Line 6: T_TRUE [ true ]
Line 6: T_NOT_EQUALS [ != ]
Line 6: T_LPAREN [ ( ]
Line 6: T_FALSE [ false ]
Line 6: T_NOT_EQUALS [ != ]
Line 6: T_LPAREN [ ( ]
Line 6: T_ID [ a ]
Line 6: T_DOUBLE_EQUALS [ == ]
Line 6: T_ID [ a ]
Line 6: T_RPAREN [ ) ]
```

Test 6: Type declaration Error

The compiler takes in the source code and creates the tokens however the parser realizes that a certain part in the source code doesn't agree with the grammer so it recognizes that and throws an error. Inside the error statement information regarding the error is displayed as well including the line number in which the error is located, the token the parser found, and lastly the token the parser was expecting. In this case the parser notices that after int the grammer is expecting a character between a-z as the identifier but instead it gets a digit so it throws an error

Output

Test program

```
{
        int 7
        a = 3
} $
```

Error

```
Parsing Error on line 2: Found T_DIGIT, expected T_ID.
```

Test 7: Boolean Error

The compiler takes in the source code and creates the tokens however the parser realizes that a certain part in the source code doesn't agree with the grammer so it recognizes that and throws an error. Inside the error statement information regarding the error is displayed as well including the line number in which the error is located, the token the parser found, and lastly the token the parser was expecting. In this case the parser recogzines that only double equal (==) are allowed in boolean expressions so it throws an error.

Output

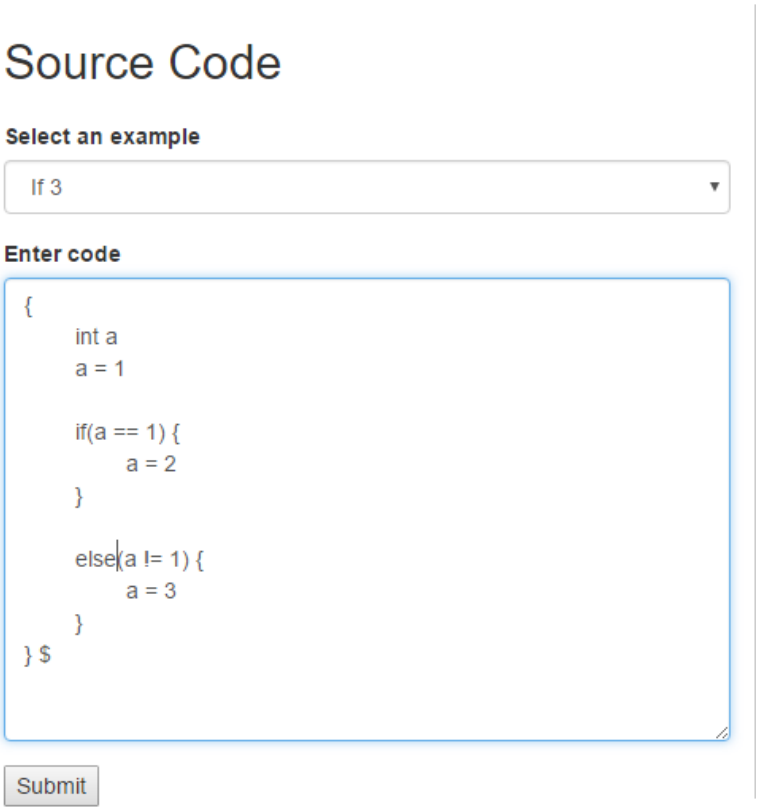
Test program

```
{
        int a
        a = 4
        if (a = 4) {
                print("hello world")
        }
} $
```

Error

```
Parsing Error on line 4: T_SINGLE_EQUALS is not a valid boolean operator.
```

Test 8: Lexeme not in the Grammer E



The compiler takes in the source code and creates the tokens however the parser realizes that a certain part in the source code doesn't agree with the grammer so it recognizes that and throws an error. Inside the error statement information regarding the error is displayed as well including the line number in which the error is located, the token the parser found, and lastly the token the parser was expecting. In this example the parser recognizes that else is not a keyword in the grammer so it throws an error.

Output

Test program

```
{
        int a
        a = 1

        if(a == 1) {
                a = 2
        }

        else(a != 1) {
                a = 3
        }
} $
```

Error

```
Lexical Error on line 9: else is not a valid lexeme.
```

Test 9: Missing Brace/Parenthesis Error

The compiler takes in the source code and creates the tokens however the parser realizes that a certain part in the source code doesn't agree with the grammer so it recognizes that and throws an error. Inside the error statement information regarding the error is displayed as well including the line number in which the error is located, the token the parser found, and lastly the token the parser was expecting. In this example the Right brace at the bottom of the code is missing so the parser through an error and placed the line number to where the right brace would be expected.

Output

Test program

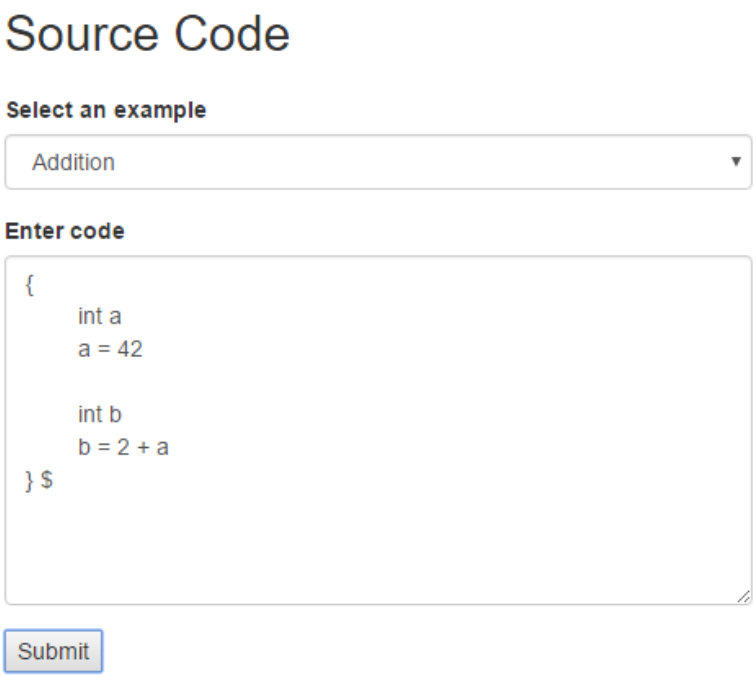
```
{
        int a
        a = 4

        int b
        b = 2 + a
$
```

Error

```
Parsing Error on line 7: Found T_EOF, expected T_RBRACE.
```

Test 10: Integer over digit Error



The compiler takes in the source code and creates the tokens however the parser realizes that a certain part in the source code doesn't agree with the grammer so it recognizes that and throws an error. Inside the error statement information regarding the error is displayed as well including the line number in which the error is located, the token the parser found, and lastly the token the parser was expecting. In this example a type int is declared and the value 42 is to be assinged to a however only single digits can be assigned to identifiers so the parser recogizes that and throws an error.

Test program

```
{
        int a
        a = 42

        int b
        b = 2 + a
} $
```

Error

```
Lexical Error on line 3: 42 is not a valid lexeme.
```