무료 전자 책

배우기

compiler-construction

Free unaffiliated eBook created from
Stack Overflow contributors.

#compiler-construction

1:

Examples

:

- , C, C ++, .
- **IDE** (: VSCode)
- . . .

.

- : . `Whitespace` , `Lolcode` , `Brainfuck` .
- : . `Swift` , `C++` `Python` .

, .

- (Ebook)
  - +
  - +
  - + `Coffeescript` `Rubby` `Coffeescript` `Rubby`
- **: , (The Dragon Book)**
  - , .
- ( )
  - .

: https://riptutorial.com/ko/compiler-construction/topic/6845/--

2:

. 4 2 4 .

.

- .
- (AST) .

Examples

python .

?

( ) . ['INTEGER', '178'] .

. .

:

```
int result = 100;
```

python :

```
import re                                   # for performing regex expressions

tokens = []                                 # for string tokens
source_code = 'int result = 100;'.split() # turning source code into list of words

# Loop through each source code word
for word in source_code:

    # This will check if a token has datatype decleration
    if word in ['str', 'int', 'bool']:
        tokens.append(['DATATYPE', word])

    # This will look for an identifier which would be just a word
    elif re.match("[a-z]", word) or re.match("[A-Z]", word):
        tokens.append(['IDENTIFIER', word])

    # This will look for an operator
    elif word in '*-/+%=':
        tokens.append(['OPERATOR', word])

    # This will look for integer items and cast them as a number
    elif re.match(".[0-9]", word):
        if word[len(word) - 1] == ';':
```

```
            tokens.append(["INTEGER", word[:-1]])
            tokens.append(['END_STATEMENT', ';'])
        else:
            tokens.append(["INTEGER", word])

print(tokens) # Outputs the token array
```

.

```
[['DATATYPE', 'int'], ['IDENTIFIER', 'result'], ['OPERATOR', '='], ['INTEGER', '100'],
['END_STATEMENT', ';']]
```

.

---

1. import regex library .
2. `tokens` . .
3. . `source_code` .
4. `source_code` .
5. .

   ```
   if word in ['str', 'int', 'bool']:
      tokens.append(['DATATYPE', word])
   ```

   .
6. . AST (Abstract Syntax Tree) .

   . https://repl.it/J9Hj/latest

   Simple Lexical Analyzer . .

?

(AST) . .

:

```
[['DATATYPE', 'int'], ['IDENTIFIER', 'result'], ['OPERATOR', '='], ['INTEGER', '100'],
['END_STATEMENT', ';']]
```

'python3' :

```
ast = { 'VariableDecleration': [] }

tokens = [ ['DATATYPE', 'int'], ['IDENTIFIER', 'result'], ['OPERATOR', '='],
           ['INTEGER', '100'], ['END_STATEMENT', ';'] ]

# Loop through the tokens and form ast
for x in range(0, len(tokens)):

    # Create variable for type and value for readability
    token_type  = tokens[x][0]
    token_value = tokens[x][1]

    # This will check for the end statement which means the end of var decl
    if token_type == 'END_STATEMENT': break

    # This will check for the datatype which should be at the first token
    if x == 0 and token_type == 'DATATYPE':
        ast['VariableDecleration'].append( {'type': token_value} )

    # This will check for the name which should be at the second token
    if x == 1 and token_type == 'IDENTIFIER':
        ast['VariableDecleration'].append( {'name': token_value} )

    # This will check to make sure the equals operator is there
    if x == 2 and token_value == '=': pass

    # This will check for the value which should be at the third token
    if x == 3 and token_type == 'INTEGER' or token_type == 'STRING':
        ast['VariableDecleration'].append( {'value': token_value} )

print(ast)
```

.

```
{'VariableDecleration': [{'type': 'int'}, {'name': 'result'}, {'value': '100'}]}
```

( ) `type` , `name` `value` .

-

1. AST `ast` .
2. `token` .
3. AST .
4. .
5. .

```
if x == 0 and token_type == 'DATATYPE':
```

```
    ast['VariableDecleration'].append( {'type': token_value} )
```

AST . `VariableDecleration` AST .

. https://repl.it/J9IT/latest

: https://riptutorial.com/ko/compiler-construction/topic/10816/--

S. No		Contributors
1		Community, RyanM, TriskalJM
2		RyanM