

NAM-Assignment2

Command Line

```
1 python runner.py example.t
```

Result

```
● dnam@Nadineui-MacBookAir-8 topic-06-grammar % python runner.py example.t
dnam@kent.edu
```

```
1 homework = True;
2 if (homework == True) {
3     dnam;
4     print _kentid_;
5 }
```

tokenizer

```
patterns = [ ]
```

```
patterns = [
    ### Added for Kent ID statement
    [r"dnam", "dnam"],
    [r"True", "True"],
    [r"False", "False"],
    ####
]
```

```
1 patterns = [
2     ### Added for Kent ID statement
3     [r"dnam", "dnam"],
```

```

4     [r"True", "True"],
5     [r"False", "False"],
6     #####
7     [r"print", "print"],
8     [r"if", "if"],
9     [r"else", "else"],
10    [r"while", "while"],
11    [r"continue", "continue"],
12    [r"break", "break"],
13    [r"return", "return"],
14    [r"assert", "assert"],
15    [r"and", "&&"],
16    [r"or", "||"],
17    [r"not", "!"],
18    [r"\d*\.\d+|\d+\.\d*|\d+", "number"],
19    [r"[a-zA-Z_][a-zA-Z0-9_]*", "identifier"], # identifiers
20    [r"\+", "+"],
21    [r"\-", "-"],
22    [r"\*", "*"],
23    [r"\/", "/"],
24    [r"\(", "("],
25    [r"\)", ")"],
26    [r"\)", ")"],
27    [r"==", "=="],
28    [r"!=", "!="],
29    [r"<=", "<="],
30    [r">=", ">="],
31    [r"<", "<"],
32    [r">", ">"],
33    [r"\=", "="],
34    [r"\;", ";"],
35    [r"&\&", "&&"],
36    [r"\\|\\", "||"],
37    [r"\\!", "!"],
38    [r"\\{", "{"],
39    [r"\\}", "}"],
40    [r"\\[", "["],
41    [r"\\]", ""]],
42    [r"\\. ", "."],
43    [r"\\s+", "whitespace"],
44    [r".", "error"]
45 ]

```

parser

def parse_factor(tokens):

```
### Added for Kent ID statement
# Change the boolean tokens to numeric values
if token["tag"] == "True":
    return {"tag": "number", "value": 1}, tokens[1:]
if token["tag"] == "False":
    return {"tag": "number", "value": 0}, tokens[1:]
###
```

```
1  def parse_factor(tokens):
2      """
3      factor = <number> | <identifier> | "(" expression ")" | "!" expression
4      | "-" expression
5      """
6      token = tokens[0]
7
8      ### Added for Kent ID statement
9      # Change the boolean tokens to numeric values
10     if token["tag"] == "True":
11         return {"tag": "number", "value": 1}, tokens[1:]
12     if token["tag"] == "False":
13         return {"tag": "number", "value": 0}, tokens[1:]
14     ###
15     if token["tag"] == "number":
16         return {"tag": "number", "value": token["value"]}, tokens[1:]
17     if token["tag"] == "identifier":
18         return {"tag": "identifier", "value": token["value"]}, tokens[1:]
19     if token["tag"] == "(":
20         ast, tokens = parse_expression(tokens[1:])
21         assert tokens[0]["tag"] == ")", f"Expected ')' but got {tokens[0]}"
22         return ast, tokens[1:]
23     if token["tag"] == "!":
24         ast, tokens = parse_expression(tokens[1:])
25         return {"tag": "!", "value": ast}, tokens
26     if token["tag"] == "-":
27         ast, tokens = parse_expression(tokens[1:])
28         return {"tag": "negate", "value": ast}, tokens
29     raise Exception(f"Unexpected token '{token['tag']}' at position {token['position']}")
```

```
def parse_statement(tokens):
```

```
    ### Added for Kent ID statement
    # AST {"tag": "dnam"}
    if tag == "dnam":
        return {"tag": "dnam"}, tokens[1:]
    ###
```

```
1  def parse_statement(tokens):
2      """
3      statement = statement_block | if_statement | while_statement |
4      print_statement | assignment_statement
5      """
6      tag = tokens[0]["tag"]
7      if tag == "{":
8          return parse_statement_block(tokens)
9      if tag == "if":
10         return parse_if_statement(tokens)
11     if tag == "while":
12         return parse_while_statement(tokens)
13     if tag == "print":
14         return parse_print_statement(tokens)
15     ### Added for Kent ID statement
16     # AST {"tag": "dnam"}
17     if tag == "dnam":
18         return {"tag": "dnam"}, tokens[1:]
19     ###
20     return parse_assignment_statement(tokens)
```

evaluator

```
def evaluate(ast, environment={}):
```

```

### Added for Kent ID statement
if ast["tag"] == "dnam":
    environment["_kentid_"] = "dnam@kent.edu"
    return None
###

```

```

1  def evaluate(ast, environment={}):
2      global printed_string
3      if ast["tag"] == "program":
4          last_value = None
5          for statement in ast["statements"]:
6              value = evaluate(statement, environment)
7              last_value = value
8          return last_value
9      if ast["tag"] == "block":
10         for statement in ast["statements"]:
11             _ = evaluate(statement, environment)
12     if ast["tag"] == "print":
13         value = evaluate(ast["value"], environment)
14         s = str(value)
15         print(s)
16         printed_string = s
17         return None
18     if ast["tag"] == "if":
19         condition_value = evaluate(ast["condition"], environment)
20         if condition_value:
21             evaluate(ast["then"], environment)
22         else:
23             if ast["else"]:
24                 evaluate(ast["else"], environment)
25         return None
26     if ast["tag"] == "while":
27         while evaluate(ast["condition"], environment):
28             evaluate(ast["do"], environment)
29         return None
30     if ast["tag"] == "assign":
31         target = ast["target"]
32         assert target["tag"] == "identifier"
33         identifier = target["value"]
34         assert type(identifier) is str
35         value = evaluate(ast["value"], environment)

```

```

36         environment[identifier] = value
37     if ast["tag"] == "number":
38         return ast["value"]
39     if ast["tag"] == "identifier":
40         if ast["value"] in environment:
41             return environment[ast["value"]]
42         parent_environment = environment
43         while "$parent" in parent_environment:
44             parent_environment = environment["$parent"]
45             if ast["value"] in parent_environment:
46                 return parent_environment[ast["value"]]
47         raise Exception(f"Value [{ast["value"]} not found in environment
{environment}].")
48     if ast["tag"] in ["+", "-", "*", "/"]:
49         left_value = evaluate(ast["left"], environment)
50         right_value = evaluate(ast["right"], environment)
51         if ast["tag"] == "+":
52             return left_value + right_value
53         if ast["tag"] == "-":
54             return left_value - right_value
55         if ast["tag"] == "*":
56             return left_value * right_value
57         if ast["tag"] == "/":
58             return left_value / right_value
59     if ast["tag"] == "negate":
60         value = evaluate(ast["value"], environment)
61         return -value
62     if ast["tag"] == "&&":
63         left_value = evaluate(ast["left"], environment)
64         right_value = evaluate(ast["right"], environment)
65         return left_value and right_value
66     if ast["tag"] == "||":
67         left_value = evaluate(ast["left"], environment)
68         right_value = evaluate(ast["right"], environment)
69         return left_value or right_value
70     if ast["tag"] == "!":
71         value = evaluate(ast["value"], environment)
72         return not value
73     if ast["tag"] == "<":
74         left_value = evaluate(ast["left"], environment)
75         right_value = evaluate(ast["right"], environment)
76         return left_value < right_value
77     if ast["tag"] == ">":
78         left_value = evaluate(ast["left"], environment)
79         right_value = evaluate(ast["right"], environment)
80         return left_value > right_value

```

```

81     if ast["tag"] == "<=":
82         left_value = evaluate(ast["left"], environment)
83         right_value = evaluate(ast["right"], environment)
84         return left_value <= right_value
85     if ast["tag"] == ">=":
86         left_value = evaluate(ast["left"], environment)
87         right_value = evaluate(ast["right"], environment)
88         return left_value >= right_value
89     if ast["tag"] == "==":
90         left_value = evaluate(ast["left"], environment)
91         right_value = evaluate(ast["right"], environment)
92         return left_value == right_value
93     if ast["tag"] == "!=":
94         left_value = evaluate(ast["left"], environment)
95         right_value = evaluate(ast["right"], environment)
96         return left_value != right_value
97     ### Added for Kent ID statement
98     if ast["tag"] == "dnam":
99         environment["_kentid_"] = "dnam@kent.edu"
100     return None
101     ###

```

Problem

Errors kept appearing during parsing, so I needed a way to exit the function when encountering
 }

```

1 Exception: Unexpected token '}' at position n.

```

Error Handling

def parse_statement_block(tokens): in parser.py

```

# To avoid errors,
# this loop skips semicolons and continues parsing the next statement
while tokens[0]["tag"] == ";":
    tokens = tokens[1:]
    ### Added for Kent ID statement
    # Stop parsing statements when "}" encountered
    if tokens[0]["tag"] == "}":
        break
    ###

```

```

1  def parse_statement_block(tokens):
2      """
3      statement_block = "{" statement { ";" statement } "}"
4      """
5      ast = {"tag": "block", "statements": []}
6      assert tokens[0]["tag"] == "{", f"Expected '{{', got {tokens[0]}"
7      tokens = tokens[1:]
8      if tokens[0]["tag"] != "}":
9          statement, tokens = parse_statement(tokens)
10         ast["statements"].append(statement)
11
12     # To avoid errors,
13     # this loop skips semicolons and continues parsing the next statement
14     while tokens[0]["tag"] == ";":
15         tokens = tokens[1:]
16         ### Added for Kent ID statement
17         # Stop parsing statements when "}" encountered
18         if tokens[0]["tag"] == "}":
19             break
20         ###
21         statement, tokens = parse_statement(tokens)
22         ast["statements"].append(statement)
23
24     assert tokens[0]["tag"] == "}", f"Expected '}}', got {tokens[0]}"
25     return ast, tokens[1:]

```