NAM-Assignment2

Command Line

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

Result

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

```
homework = True;
if (homework == True) {
    dnam;
    print _kentid_;
}
```

---

tokenizer

patterns = [ ]


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

```
    [r"True", "True"],
    [r"False", "False"],
    ####
    [r"print","print"],
    [r"if","if"],
    [r"else","else"],
    [r"while","while"],
    [r"continue","continue"],
    [r"break","break"],
    [r"return","return"],
    [r"assert","assert"],
    [r"and","&&"],
    [r"or","||"],
    [r"not","!"],
    [r"\d*\.\d+|\d+\.\d*|\d+", "number"],
    [r"[a-zA-Z_][a-zA-Z0-9_]*", "identifier"],  # identifiers
    [r"\+", "+"],
    [r"\-", "-"],
    [r"\*", "*"],
    [r"\/", "/"],
    [r"\(", "("],
    [r"\)", ")"],
    [r"\)", ")"],
    [r"==", "=="],
    [r"!=", "!="],
    [r"<=", "<="],
    [r">=", ">="],
    [r"<", "<"],
    [r">", ">"],
    [r"\=", "="],
    [r"\;", ";"],
    [r"\&\&", "&&"],
    [r"\|\|", "||"],
    [r"\!", "!"],
    [r"\{", "{"],
    [r"\}", "}"],
    [r"\[", "["],
    [r"\]", "]"],
    [r"\.", "."],
    [r"\s+","whitespace"],
    [r".","error"]
]
```

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:]
###
```

```
def parse_factor(tokens):
    """
    factor = <number> | <identifier> | "(" expression ")" | "!" expression | "-" expression
    """
    token = tokens[0]

    ### 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:]
    ###
    if token["tag"] == "number":
        return {"tag": "number", "value": token["value"]}, tokens[1:]
    if token["tag"] == "identifier":
        return {"tag": "identifier", "value": token["value"]}, tokens[1:]
    if token["tag"] == "(":
        ast, tokens = parse_expression(tokens[1:])
        assert tokens[0]["tag"] == ")", f"Expected ')' but got {tokens[0]}"
        return ast, tokens[1:]
    if token["tag"] == "!":
        ast, tokens = parse_expression(tokens[1:])
        return {"tag": "!", "value": ast}, tokens
    if token["tag"] == "-":
        ast, tokens = parse_expression(tokens[1:])
        return {"tag": "negate", "value": ast}, tokens
    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:]
###
```

```
def parse_statement(tokens):
    """
    statement = statement_block | if_statement | while_statement | print_statement | assignment_statement
    """
    tag = tokens[0]["tag"]
    if tag == "{":
        return parse_statement_block(tokens)
    if tag == "if":
        return parse_if_statement(tokens)
    if tag == "while":
        return parse_while_statement(tokens)
    if tag == "print":
        return parse_print_statement(tokens)
    ### Added for Kent ID statement
    # AST {"tag": "dnam"}
    if tag == "dnam":
        return {"tag": "dnam"}, tokens[1:]
    ###
    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
###
```

```
def evaluate(ast, environment={}):
    global printed_string
    if ast["tag"] == "program":
        last_value = None
        for statement in ast["statements"]:
            value = evaluate(statement, environment)
            last_value = value
        return last_value
    if ast["tag"] == "block":
        for statement in ast["statements"]:
            _ = evaluate(statement, environment)
    if ast["tag"] == "print":
        value = evaluate(ast["value"], environment)
        s = str(value)
        print(s)
        printed_string = s
        return None
    if ast["tag"] == "if":
        condition_value = evaluate(ast["condition"], environment)
        if condition_value:
            evaluate(ast["then"], environment)
        else:
            if ast["else"]:
                evaluate(ast["else"], environment)
        return None
    if ast["tag"] == "while":
        while evaluate(ast["condition"], environment):
            evaluate(ast["do"], environment)
        return None
    if ast["tag"] == "assign":
        target = ast["target"]
        assert target["tag"] == "identifier"
        identifier = target["value"]
        assert type(identifier) is str
        value = evaluate(ast["value"],environment)
```

```
        environment[identifier] = value
    if ast["tag"] == "number":
        return ast["value"]
    if ast["tag"] == "identifier":
        if ast["value"] in environment:
            return environment[ast["value"]]
        parent_environment = environment
        while "$parent" in parent_environment:
            parent_environment = environment["$parent"]
            if ast["value"] in parent_environment:
                return parent_environment[ast["value"]]
        raise Exception(f"Value [{ast["value"]}] not found in environment
{environment}.")
    if ast["tag"] in ["+", "-", "*", "/"]:
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        if ast["tag"] == "+":
            return left_value + right_value
        if ast["tag"] == "-":
            return left_value - right_value
        if ast["tag"] == "*":
            return left_value * right_value
        if ast["tag"] == "/":
            return left_value / right_value
    if ast["tag"] == "negate":
        value = evaluate(ast["value"], environment)
        return -value
    if ast["tag"] == "&&":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value and right_value
    if ast["tag"] == "||":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value or right_value
    if ast["tag"] == "!":
        value = evaluate(ast["value"], environment)
        return not value
    if ast["tag"] == "<":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value < right_value
    if ast["tag"] == ">":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value > right_value
```

```
    if ast["tag"] == "<=":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value <= right_value
    if ast["tag"] == ">=":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value >= right_value
    if ast["tag"] == "==":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value == right_value
    if ast["tag"] == "!=":
        left_value = evaluate(ast["left"], environment)
        right_value = evaluate(ast["right"], environment)
        return left_value != right_value
    ### Added for Kent ID statement
    if ast["tag"] == "dnam":
        environment["_kentid_"] = "dnam@kent.edu"
        return None
    ###
```

Problem

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

```
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
    ###
```

```
def parse_statement_block(tokens):
    """
    statement_block = "{" statement { ";" statement } "}"
    """
    ast = {"tag": "block", "statements": []}
    assert tokens[0]["tag"] == "{", f"Expected '{{', got {tokens[0]}"
    tokens = tokens[1:]
    if tokens[0]["tag"] != "}":
        statement, tokens = parse_statement(tokens)
        ast["statements"].append(statement)

    # 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
        ###
        statement, tokens = parse_statement(tokens)
        ast["statements"].append(statement)

    assert tokens[0]["tag"] == "}", f"Expected '}}', got {tokens[0]}"
    return ast, tokens[1:]
```