|  |
| --- |
| Team 10 |
| **Compiler Term-Project #2** |
| The implementation of a bottom-up syntax analyzer |

|  |  |
| --- | --- |
| Date | June 20, 2020 |
| Instructor | Hyosu Kim |
| Team | #10 |
| Members | Heesang Ro (20145001)  Junhyuck Woo (20145337) |



# INDEX

[INDEX 1](#_Toc43064540)

[SPECIFICATIONS 3](#_Toc43064541)

[Modified part in specifications 3](#_Toc43064542)

[FIRST SET 4](#_Toc43064543)

[FOLLOW SET 5](#_Toc43064544)

[NFA (Non-deterministic Finite Automata) 6](#_Toc43064545)

[1. Overall Graph 6](#_Toc43064546)

[DFA (Deterministic Finite Automata) 7](#_Toc43064547)

[1. NFA with numbering 7](#_Toc43064548)

[2. Subset Construction 8](#_Toc43064549)

[SLR PARSING TABLE 9](#_Toc43064550)

[CODE ALGORITHM 10](#_Toc43064551)

[TROUBLE & SOLUTION 11](#_Toc43064552)

[1. Longest matching 11](#_Toc43064553)

[2. Period 12](#_Toc43064554)

[3. Error 13](#_Toc43064555)

[IMPLEMENTATION 14](#_Toc43064556)

[1. Definition of Tokens, Alphabet 14](#_Toc43064557)

[2. ID-DFA 15](#_Toc43064558)

[3. INTEGER - DFA 16](#_Toc43064559)

[4. FLOAT - DFA 17](#_Toc43064560)

[5. LITERAL - DFA 18](#_Toc43064561)

[6. Other Tokens 19](#_Toc43064562)

[7. Other - File I/O 20](#_Toc43064563)

[TEST CASES & RESULT 21](#_Toc43064564)

[1. Correct Test Code 21](#_Toc43064565)

[2. Error Test Code 22](#_Toc43064566)

[APPENDIX 24](#_Toc43064567)

[1. First Set 24](#_Toc43064568)

[2. Follow Set 25](#_Toc43064569)

[3. NFA to DFA 26](#_Toc43064570)

[4. Transition Table 27](#_Toc43064571)

# SPECIFICATIONS

|  |
| --- |
| **CFG G:**   1. CODE → VDECL CODE | FDECL CODE | ε 2. VDECL → vtype id semi | vtype ASSIGN semi 3. ASSIGN → id assign RHS 4. FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace 5. ARG → vtype id MOREARGS | ε 6. MOREARGS → comma vtype id MOREARGS | ε 7. BLOCK → STMT BLOCK | ε 8. STMT → VDECL | ASSIGN semi 9. STMT → if lparen COND rparen lbrace BLOCK rbrace ELSE 10. STMT → while lparen COND rparen lbrace BLOCK rbrace 11. STMT → for lparen ASSIGN semi COND semi ASSIGN rparen lbrace BLOCK rbrace 12. ELSE → else lbrace BLOCK rbrace | ε 13. RHS → EXPR | literal 14. EXPR → TERM addsub EXPR | TERM 15. TERM → FACTOR multdiv TERM | FACTOR 16. FACTOR → lparen EXPR rparen | id | num | float 17. COND → FACTOR comp FACTOR 18. RETURN → return FACTOR semi   **Terminals**   1. **vtype** for the types of variables and functions 2. **num** for signed integers 3. **float** for floating-point numbers 4. **literal** for literal strings 5. **id** for the identifiers of variables and functions 6. **if**, **else**, **while**, **for** and **return** for if, else, while, for and return statements respectively 7. **addsub** for + and - arithmetic operators 8. **multdiv** for \* and / arithmetic operators 9. **assign** for assignment operators 10. **comp** for comparison operators 11. **semi** and **comma** for semicolons and commas respectively 12. **lparen**, **rparen**, **lbrace**, and **rbrace** for (, ), {, and } respectively   **Non-terminals**  CODE, VDECL, FDECL, ARG, MOREARGS, BLOCK, STMT, ASSIGN, RHS, EXPR, TERM, FACTOR, COND, RETURN  **Start symbol**  CODE |

## Modified part in specifications

In the CFG, we add one more line.

00: GOAL -> CODE

# FIRST SET

This is the result of First Set. You can check our handwriting version at the [appendix 1](#_1._First_Set).

|  |
| --- |
| First(RETURN) = {return}  First(FACTOR) = {lparen, id, num, float}  First(COND) = {lparen, id, num, float}  Fist(TERM) = {lparen, id, num, float}  First(EXPR) = {lparen, id, num, float}  First(RHS) = {lparen, id, num, float, literal}  First(ELSE) = {else, ε}  First (VDECL) = {vtype}  Fist (ASSIGN) = {id}  First (STMT) = {for, while, if, vtype, id}  First (BLOCK) = {for, while, if, vtype, id, ε}  First (MOREARGS) = {comma, ε}  First (ARG) = {vtype, ε}  First (FDECL) = {vtype}  First (CODE) = {vtype, ε}  First (GOAL) = {vtype, ε} |

# FOLLOW SET

This is the result of Follow Set. You can check our handwriting version at the [appendix 2](#_2._Follow_Set).

|  |
| --- |
| Follow (GOAL) = {$}  Follow (CODE) = {$}  Follow (VDECL) = {ε, vtype, if, while, for, id}  Follow (ASSIGN) = {semi, rparen}  Follow (FDECL) = {ε, vtype}  Follow (ARG) = {rparen}  Follow (MOREARGS) = {rparen}  Follow (BLOCK) = {rbrace, return}  Follow (STMT) = {ε, if, while, ofr, vtype, id}  Follow (ELSE) = {ε, if, while, ofr, vtype, id}  Follow (RHS) = {semi, rparen}  Follow (EXPR) = {semi, rparen}  Follow (TERM) = {addsub, rparen, semi}  Follow (FACTOR) = {semi, comp, multdiv, rparen, addsub}  Follow (COND) = {semi, rparen}  Follow (RETURN) = {rbrace} |

# NFA (Non-deterministic Finite Automata)

Our team drew the NAF (Non-deterministic Finite Automata) using flow chart drawing tool[[1]](#footnote-1). The graph is too big, so it is hard to see the detail, so we included the drawing file in the “Handwriting” folder, if you want please check the directory.

## 1. Overall Graph

텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

# DFA (Deterministic Finite Automata)

DFA is generated using the NFA which we built. At first, we attached the number on each node and derived the DFA using subset (powerset) construction algorithm. You can check the handwriting version at the [appendix 3](#_3._NFA_to).

## 1. NFA with numbering

This image also can be found at the “Handwriting” directory.

텍스트, 지도, 그리기이(가) 표시된 사진

자동 생성된 설명

## 2. Subset Construction

|  |
| --- |
|  |

## 3. Graph

# SLR PARSING TABLE

# CODE ALGORITHM

# TROUBLE & SOLUTION

## 1. Longest matching

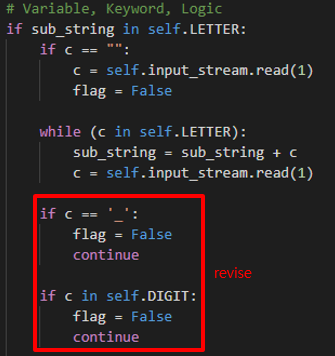
* Trouble

When we tested identifier that includes keyword and variable (e.g. if, while, int, char), longest matching didn’t work well.

|  |  |  |
| --- | --- | --- |
| Input | Expected result | Result |
| 검은색, 앉아있는, 화면, 전화이(가) 표시된 사진  자동 생성된 설명 | ……  VARIABLE int  ID int\_ABC  ASSIGN =  …… | ……    …… |

* Solution

We add more if condition in variable, keyword, logic part so that our program can do longest matching.



* Result

|  |  |  |
| --- | --- | --- |
| Input | Previous result | Revised result |
|  | ……    …… |  |

## 2. Period

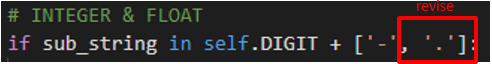
* Trouble

When we tested . (Period) without any prefix, our program is finished that line.

|  |  |  |
| --- | --- | --- |
| Input | Expected result | Result |
|  | Line2 error! | Program is finished unexpected. |

* Solution

At first, our program didn’t include the condition that what we did if the first substring is . (Period). So, we add . (Period) condition in our program.



* Result

|  |  |  |
| --- | --- | --- |
| Input | Previous result | Revised result |
|  | Program is finished unexpected. |  |

## 3. Error

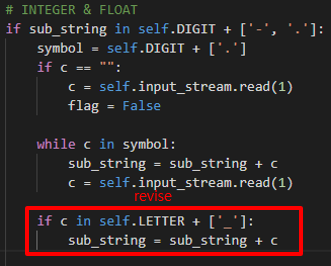
* Trouble

When we tested wrong inputs like 33abc, 3.3test, our program didn’t get the error.

|  |  |  |
| --- | --- | --- |
| Input | Expected result | Result |
| 어두운, 검은색, 시계, 오렌지이(가) 표시된 사진  자동 생성된 설명 | Line2 error! | ……    …… |

* Solution

We add more if condition in int, float part so that our program can do longest matching.



* Result

|  |  |  |
| --- | --- | --- |
| Input | Previous result | Revised result |
|  | ……    …… |  |

# IMPLEMENTATION

Before explaining our works, we will introduce the developing environment and how to manage our project, please check the appendix-2.

|  |
| --- |
| **Language**   * Python3 (version: 3.7.4)   **Operating System**   * macOS Catalina * Windows 10   **IDE (Integrated Development Environment)**   * Visual Studio Code (version: 1.45.0) * PyCharm (version: 3.9.4)   **Project Management**   * Git (version: 2.24.2) * Git-Hub |

We defined the four tokens: ID, INTEGER, FLOAT, LITERAL as function, and the others are defined as a list type variable. In addition, we defined five symbols; LETTER, SYMBOL, ZERO, NON\_ZERO and DIGIT to improve the productivity and readability.

## 1. Definition of Tokens, Alphabet

We defined the four tokens: ID, INTEGER, FLOAT, LITERAL as function, and the others are defined as a list type variable. In addition, we defined five symbols; LETTER, SYMBOL, ZERO, NON\_ZERO and DIGIT to improve the productivity and readability.

스크린샷이(가) 표시된 사진

자동 생성된 설명

## 2. ID-DFA

스크린샷이(가) 표시된 사진

자동 생성된 설명

This part of code is for ID DFA that decides whether the input\_string is ID or not. This method has 7 states, and the start state is 0. In a for-loop, it works exactly the same as the ID DFA that was designed in the previous page. This method reads input\_string in order, and a recent state is changed by input\_string. If it is finished in the final states, which are 1 to 6, the input\_string will be accepted. Otherwise, the input\_string will be denied, and it means the input\_string isn’t ID.

## 3. INTEGER - DFA

스크린샷, 앉아있는, 테이블, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

This part of code is for INT DFA that decides whether the input\_string is int or not. This method has 6 states, and the start state is T0. In a for-loop, it works exactly the same as the INT DFA that was designed in the previous page. This method reads input\_string in order, and a recent state is changed by input\_string. If it is finished in the final states, which are state2, state3, state4, and state5, the input\_string will be accepted. Otherwise, the input\_string will be denied, and it means input\_string isn’t int.

## 4. FLOAT - DFA

스크린샷, 앉아있는, 테이블, 노트북이(가) 표시된 사진

자동 생성된 설명전화이(가) 표시된 사진

자동 생성된 설명

This part of the code is for FLOAT DFA that decides whether the input\_string is float or not. This method has 10 states, and the start state is T0. In a for-loop, it works exactly the same as the FLOAT DFA that was designed in the previous page. This method reads input\_string in order, and a recent state is changed by input\_string. If it is finished in the final states, which are state7 and state8, the input\_string will be accepted. Otherwise, the input\_string will be denied, and it means input\_string isn’t a float.

## 5. LITERAL - DFA

스크린샷이(가) 표시된 사진

자동 생성된 설명

This part of code is for LITERAL DFA that decides whether the input\_string is a string or not. This method has 7 states, and the start state is 0. In a for-loop, it works exactly the same as the LITERAL DFA that was designed in the previous page. This method reads input\_string in order, and a recent state is changed by input\_string. If it is finished in the final state, which is 2, the input\_string will be accepted. Otherwise, the input\_string will be denied, and it means input\_string isn’t a string.

## 6. Other Tokens

The software checks the other tokens following matching processing.

|  |  |
| --- | --- |
| * Variable, Keyword, Logic   텍스트, 스크린샷이(가) 표시된 사진  자동 생성된 설명   * ASSIGN, COMPARISON   스크린샷이(가) 표시된 사진  자동 생성된 설명 | * BRACE, PAREN, TERM, COMMA, OPERATOR, COMPARISON   텍스트, 스크린샷이(가) 표시된 사진  자동 생성된 설명 |

* Subtract

그리기이(가) 표시된 사진

자동 생성된 설명

## 7. Other - File I/O

* Read File

|  |  |
| --- | --- |
| 쥐고있는, 남자이(가) 표시된 사진  자동 생성된 설명 | 시계, 측정기, 검은색, 표지판이(가) 표시된 사진  자동 생성된 설명 |

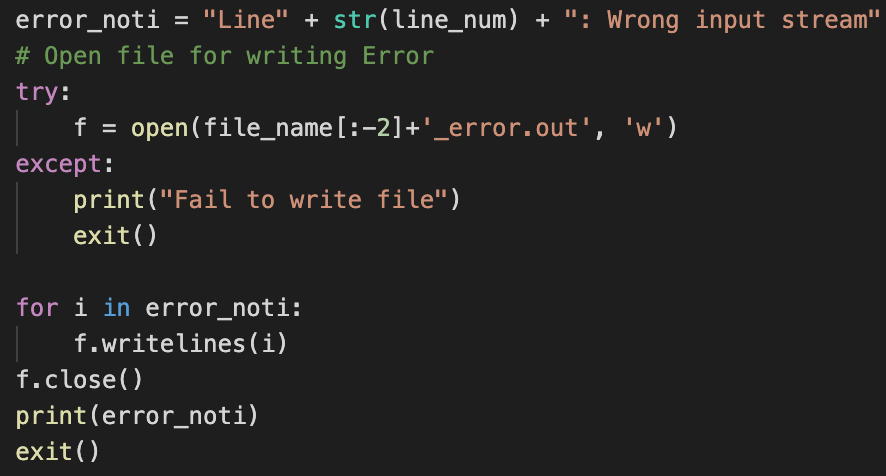
* Write File

텍스트이(가) 표시된 사진

자동 생성된 설명

* Error Case

If the code is written as out of correct grammar, the software returns the error message and creates the error message file as followed format FIEL\_NAME\_error.out. There are 5 points where check the error; each of them use the same code, so we attached a single picture.



# TEST CASES & RESULT

## 1. Correct Test Code

|  |  |
| --- | --- |
| **Input** | **Result** |
| **텍스트, 화면, 테이블이(가) 표시된 사진  자동 생성된 설명** | 텍스트이(가) 표시된 사진  자동 생성된 설명텍스트이(가) 표시된 사진  자동 생성된 설명 |

## 2. Error Test Code

* In our lexical analyzer, we don’t allow to use the character ‘$’.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘=!’ isn’t correct comparison.

|  |  |
| --- | --- |
| **Input** | **Result** |
|  |  |

* ‘.0’ isn’t correct float.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색, 어두운, 화면, 오렌지이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘1.10’ isn’t correct float. In our lexical analyzer, the right side of a decimal point must be a single digit 0 or a non-empty sequence terminating with a non-zero digit.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색, 어두운, 앉아있는, 측정기이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘a.a’ isn’t correct input.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색, 오렌지, 어두운, 화면이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘12people’ isn’t correct ID. ID should be started with English letter or ‘\_’.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 어두운, 사진, 오렌지, 모니터이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘013’ isn’t corret int.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색, 어두운, 오렌지, 측정기이(가) 표시된 사진  자동 생성된 설명 |  |

* ‘-0’ isn’t correct int.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색, 모니터, 어두운, 화면이(가) 표시된 사진  자동 생성된 설명 |  |

* Literal string should be terminated with a symbol “.

|  |  |
| --- | --- |
| **Input** | **Result** |
| 검은색이(가) 표시된 사진  자동 생성된 설명 |  |

# APPENDIX

## 1. First Set

텍스트이(가) 표시된 사진

자동 생성된 설명

## 2. Follow Set

## 3. NFA to DFA

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

## 4. Transition Table

1. https://app.diagrams.net [↑](#footnote-ref-1)