CAU PL - LL(1) parser - External Document (1)

20224566 김동우

LL(1)파서 제작 과제의 External Document 이다

Dependency

anytree 설치법

pip3 install anytree

또는

pip install anytree

-v 옵션을 사용하지 않을때 실행방법 예시

python3 main.py test.txt

또는

python main.py test.txt

-v 옵션을 사용할때 실행방법 예시

python3 main.py -v test.txt

또는

python main.py -v test.txt

Error List - Lexer.py

error case 1 식별자의 이름이 c언어의 규칙을 따르지 않을 때

`출력되는 오류 메시지`

`(Error) Unknown token - invalid identifier(Does not follow the identifier name rules for language c) (` + invalid identifier name + `)`

`오류 메시지가 출력되는 상황`

직전 토큰이 식별자로 인식되었고 그 식별자 다음 공백이 존재하지 않으며 식별자 다음 온 문자가 연산자와 같은 문법상 가능한 문자가 아닌 경우에 식별자의 이름이 c언어의 규칙을 따르지 않는다고 판단됨

`예시 입력`

a!f := 100

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a!f := 100
ID: 1; CONST: 1; OP: 0;
(Error) Unknown token - invalid identifier(Does not follow the identifier name rules for language c) (a!f)

Result ==> a!f: invalid identifier name;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - c언어 이름 규칙을 따르지 않는 연산자는 앞쪽에서 c언어 이름 규칙을 따르는 부분까지만 detect_id()에 의해 인식되고 규칙을 따르지 않는 뒷쪽 부분들은 한글자씩 token_string에 추가되는 방식으로 구현되었으므로 a!f는 3번에 걸쳐 출력됨.

a
a!
a!f
:=
100
EOF

error case 2 !나 @, 한글 등 문법상 등장할 수 없는 문자가 등장했을 때

`출력되는 오류 메시지`

`(Error) Unknown token - invalid character(!, @, etc.) or string with invalid character (` + invalid character or string + `)`

`오류 메시지가 출력되는 상황`

한글이나 !, @, 같이 문법상 등장할 수 없는 문자 또는 문자열이 등장한 경우 - error case 1의 오류와 error case 2의 오류가 동시에 있다면 error case 1의 오류만 출력된다.

`예시 입력`

프로그래밍언어론 := 100

`v위의 예시 입력에 대한 -옵션이 없을 때의 예시 출력`

프로그래밍언어론 := 100
ID: 0; CONST: 1; OP: 0;

(Error) Unknown token - invalid character(!, @, etc.) or string with invalid character (프로그래밍언어론)

Result ==>There is no identifier

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - 규칙을 따르지 않는 문자 또는 문자열은 한글자씩 token_string에 추가되는 방식으로 구현되었으므로 프로그래밍언어론는 8번에 걸쳐 출력됨.

프
프로
프로그
프로그래
프로그래밍
프로그래밍언
프로그래밍언어
프로그래밍언어론
:=
100
EOF

error case 3 왼쪽 괄호가 식별자 직후에 등장하는 경우

`출력되는 오류 메시지`

`(Error) There is left parenthesis after identifier`

`오류 메시지가 출력되는 상황`

현재 인식된 연산자가 (이고 before_token이 TokenType.IDENT일경우

`예시 입력`

a := 10;

b := a(1+1)

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a := 10;
ID: 1; CONST: 1; OP: 0;
(OK)

b := a(1+1)
ID: 2; CONST: 2; OP: 1;
(Error) There is left parenthesis after identifier

Result ==> a: 10; b: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - 오류로 인해 파싱은 종료되었지만 Lexer는 계속 동작하므로 (이후의 부분도 출력된다.

a
:=
10
;
b
:=
a
(
1
+
1
)
EOF

error case 4 대입 연산자이후 연산자가 등장하는 경우

`출력되는 오류 메시지`

`(Error) Operator(operater or right_paren, semi_colon, assign_op) after assignment operator`

`예시 입력`

a := + 1

`오류 메시지가 출력되는 상황`

:= 다음에 +-/*;:=같은 문자들이 나오는 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a := + 1
ID: 1; CONST: 1; OP: 1;
(Error) Operator(operater or right_paren, semi_colon, assign_op) after assignment operator

Result ==> a: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

a
:=
+

1
EOF

error case 5 식별자가 숫자로 시작하는 경우

`출력되는 오류 메시지`

`(Error) Identifier starts with digit (` + Identifier which starts with dight + `)`

`예시 입력`

1a := 1 + 1

`오류 메시지가 출력되는 상황`

상수가 lexer에 의해 인식된 이후 나온 문자가 `.+-*/();:=`나 공백이 아닌 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - Result에서 1a가 invalid identifier name으로 출력된다.

1a := 1 + 1
ID: 1; CONST: 2; OP: 1;
(Error) Identifier starts with digit (1a)

Result ==> 1a: invalid identifier name;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

1a
:=
1
+
1
EOF

error case 6 식별자가 c언어의 식별자 이름 규칙을 따르지 않는 경우 중 식별자가 숫자로 시작하는 경우를 제외한 나머지 경우

`출력되는 오류 메시지`

`(Error) Unknown token - invalid identifier(Does not follow the identifier name rules for language c) (` + invalid identifier name + `)`

`예시 입력`

a! := 1 + 1

`오류 메시지가 출력되는 상황`

lexer에 의해 인식된 식별자 이후 나온 문자가 `.+-*/();:=`나 공백이 아닌 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - Result에서 1a가 invalid identifier name으로 출력된다.

a! := 1 + 1
ID: 1; CONST: 2; OP: 1;
(Error) Unknown token - invalid identifier(Does not follow the identifier name rules for language c) (a!)

Result ==> a!: invalid identifier name;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - lexer에 의해 식별자로 인식된 부분 이후의 부분은 1글자씩 token_string에 추가되도록 구현되어서 a!는 2회에 걸쳐 출력된다.

a
a!
:=
1
+
1
EOF

error case 7-1 선언(식별자에 값이 정상적으로 할당)되지 않은 식별자가 사용된 경우

→ parser의 동작이 중지되었을때만 lexer에서 처리한다.

`출력되는 오류 메시지`

`(Error) Using undeclared identifier(` + undeclared identifier + `)`

`예시 입력`

b := a

`오류 메시지가 출력되는 상황`

선언하지 않은 연산자가 사용되었거나 이전에 오류로 인해 값이 정상적으로 할당되지 않은 식별자가 사용되었을 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

b := a
ID: 2; CONST: 0; OP: 0;
(Error) Undefined variable is referenced(a)

Result ==> b: Unknown; a: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

b
:=
a
EOF

Warning List - Lexer.py

warning case 1 여러 개의 식별자가 연속해서 등장하는 경우

`출력되는 오류 메시지`

`(Warning) Continuous identifiers - ignoring identifiers (` + continouous identifiers except the first one + `)`

`오류 메시지가 출력되는 상황`

next_token과 before_token이 TokenType.IDENT일경우

`예시 입력`

a := 10;

b := 20;

c := 30;

d := a b c

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - b와 c가 무시되어 d의 결과가 a에 의해 10이 되었다.

a := 10;
ID: 1; CONST: 1; OP: 0;
(OK)

b := 20;
ID: 1; CONST: 1; OP: 0;
(OK)

c := 30;
ID: 1; CONST: 1; OP: 0;
(OK)

d := a
ID: 2; CONST: 0; OP: 0;
(Warning) Continuous identifiers - ignoring identifiers(b)
(Warning) Continuous identifiers - ignoring identifiers(c)

Result ==> a: 10; b: 20; c: 30; d: 10;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - b와 c가 무시되어 출력되지 않았다.

a
:=
10
;
b
:=
20
;
c
:=
30
;
d
:=
a
EOF

warning case 2 여러 개의 상수가 연속해서 등장하는 경우

`출력되는 오류 메시지`

`(Warning) Continuous constants - ignoring constants (` + continouous constants except the first one + `)`

`예시 입력`

a := 3.14159265358979 1 2 3 4 5 6

`오류 메시지가 출력되는 상황`

next_token과 before_token이 TokenType.CONST일경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - 1 2 3 4 5 6이 무시된 것을 확인할 수 있다.

a := 3.14159265358979

ID: 1; CONST: 1; OP: 0;
(Warning) Continuous constants - ignoring constants(1)
(Warning) Continuous constants - ignoring constants(2)
(Warning) Continuous constants - ignoring constants(3)
(Warning) Continuous constants - ignoring constants(4)
(Warning) Continuous constants - ignoring constants(5)
(Warning) Continuous constants - ignoring constants(6)

Result ==> a: 3.14159265358979;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - 1 2 3 4 5 6이 무시된 것을 확인할 수 있다.

a
:=
3.14159265358979
EOF

warning case 3 소수점이 여러 개인 소수가 등장하는 경우

`출력되는 오류 메시지`

`(Warning) Multiple decimal points - ignoring decimal points and digits after the second decimal point(` + from second decimal point to end of constant + `)`

`예시 입력`

a := 3.1415.92.65.3.5.8.9.7

`오류 메시지가 출력되는 상황`

next_token이 TokenType.CONST이고 인식된 상수 토큰에 .이 포함(소수)이며 상수토큰 다음 문자가 .일 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - .92.65.3.5.8.9.7이 무시된 것을 확인할 수 있다.

a := 3.1415
ID: 1; CONST: 1; OP: 0;
(Warning) Multiple decimal points - ignoring decimal points and digits after the second decimal point(.92.65.3.5.8.9.7)

Result ==> a: 3.1415;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - .92.65.3.5.8.9.7이 무시된 것을 확인할 수 있다.

a
:=
3.1415
EOF

warning case 4 :=가 :로 잘못 작성된 경우

`출력되는 오류 메시지`

`(Warning) Using = instead of := ==> assuming :=`

`오류 메시지가 출력되는 상황`

:가 Lexer에 의해 인식된 경우

`예시 입력`

a : 10

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - :가 :=로 처리되었다.

a := 10
ID: 1; CONST: 1; OP: 0;
(Warning) Using : instead of := ==> assuming :=

Result ==> a: 10;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - :가 :=로 처리되었다.

a
:=
10
EOF

warning case 5 :=가 =로 잘못 작성된 경우

`출력되는 오류 메시지`

`(Warning) Using = instead of := ==> assuming :=`

`오류 메시지가 출력되는 상황`

=가 Lexer에 의해 인식된 경우

`예시 입력`

a = 10

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - =가 :=로 처리되었다.

a := 10
ID: 1; CONST: 1; OP: 0;
(Warning) Using : instead of := ==> assuming :=

Result ==> a: 10;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - =가 :=로 처리되었다.

a
:=
10
EOF

warning case 6 여러 개의 연산자가 연속해서 등장하는 경우

`출력되는 오류 메시지`

`(Warning) Using multiple operators(operater or left_paren) ==> ignoring multiple operators except the first one(` + continouous operators except the first one + `)`

`예시 입력`

a := 1 +-*/ 2

`오류 메시지가 출력되는 상황`

next_token과 before_token이 TokenType.ADD_OP나 TokenType.MULT_OP중 하나인 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력` - +뒤의 -*/ 가 무시된 것을 볼 수 있다.

a := 1 + 2
ID: 1; CONST: 2; OP: 1;
(Warning) Using multiple operators(operater or left_paren) ==> ignoring multiple operators except the first one(-*/)

Result ==> a: 3;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - +뒤의 -*/ 가 무시된 것을 볼 수 있다.

a
:=
1
+
2
EOF

warning case 7-1 프로그램의 끝이 ;인 경우

→오류로 인해 Parser가 아닌 Lexer만 작동하는 상황에서만 Lexer.py에서 작동됨

`출력되는 오류 메시지`

`(Warning) There is semicolon at the end of the program ==> ignoring semicolon`

`예시 입력`

b := a! + (1 + 1);

`오류 메시지가 출력되는 상황`

오류로 인해 Lexer만 작동중이며 program의 끝이 ;인 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

b := a! + (1 + 1)
ID: 2; CONST: 2; OP: 2;
(Error) Unknown token - invalid identifier(Does not follow the identifier name rules for

language c) (a!)
(Warning) There is semicolon at the end of the program ==> ignoring semicolon

Result ==> b: Unknown; a!: invalid identifier name;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - ;가 출력되지 않은 것을 볼 수 있다.

b
:=
a
a!
+
(
1
+
1
)
EOF

Error List - Parser.py

error case 8 입력받은 소스코드가 공백으로만 이루어진 경우

→주어진 문법은 공백을 생성할 수 없다. <statement>는 소스코드에서 1개 이상 등장할 수 밖에 없으며 <statement> → <ident><assignment_op><expression> 에서 <statement>는 정상적인 상황이라면 공백으로만 이루어질 수 없다.

`출력되는 오류 메시지`

`(Error) Grammer of this LL(1) parser cannot generate empty source code`

`예시 입력`

`오류 메시지가 출력되는 상황`

소스코드로 공백이 입력된 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

ID: 0; CONST: 0; OP: 0;
(Error) Grammer of this LL(1) parser cannot generate empty source code

Result ==>There is no identifier

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

EOF

error case 9 Syntax Error - 나머지 에러들에 해당하지 않는 애러

`출력되는 오류 메시지`

`(Error) Syntax error - invalid token or invalid token sequence or missing token`

`예시 입력`

a := 1 +

`오류 메시지가 출력되는 상황`

잘못된 토큰이 사용되거나 토큰들의 순서가 잘못되었거나 문법상 등장하여야하는 종류의 토큰이 등장하지 않았을때

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a := 1 +
ID: 1; CONST: 1; OP: 1;
(Error) Syntax error - invalid token or invalid token sequence or missing token

Result ==> a: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

a
:=
1
+
EOF

error case 7-2 선언(식별자에 값이 정상적으로 할당)되지 않은 식별자가 사용된 경우

→다른 에러들과 달리 이 에러는 syntax error가 아니라 semantic error이므로 Parser가 계속 동작한다.

`출력되는 오류 메시지`

`(Error) Using undeclared identifier(` + undeclared identifier + `)`

`예시 입력`

b := a

`오류 메시지가 출력되는 상황`

선언하지 않은 연산자가 사용되었거나 이전에 오류로 인해 값이 정상적으로 할당되지 않은 식별자가 사용되었을 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

b := a
ID: 2; CONST: 0; OP: 0;
(Error) Undefined variable is referenced(a)

Result ==> b: Unknown; a: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

b
:=
a
EOF

error case 10 assignment operator가 statement에 없는 경우

`출력되는 오류 메시지`

`(Error) Missing assignment operator`

`예시 입력`

a 10

`오류 메시지가 출력되는 상황`

assignment operator가 statement에 없는 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a 10
ID: 1; CONST: 1; OP: 0;
(Error) Missing assignment operator

Result ==> a: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

a

10
EOF

error case 11 왼쪽 괄호가 없이 오른쪽 괄호가 나온 경우

`출력되는 오류 메시지`

`(Error) Missing left parenthesis`

`예시 입력`

b := 1 + 1)

`오류 메시지가 출력되는 상황`

왼쪽 괄호가 빠진 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

b := 1 + 1)
ID: 1; CONST: 2; OP: 1;
(Error) Missing left parenthesis

Result ==> b: Unknown;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력`

b
:=
1
+
1
)
EOF

Warning List - Parser.py

warning case 8 괄호가 닫히지 않은 경우

→오류로 인해 Parser가 아닌 Lexer만 작동하는 상황에서는 출력되지 않음

`출력되는 오류 메시지`

`(Warning) Missing right parenthesis ==> assuming right parenthesis at the end of statement`

`예시 입력`

a := (1 + 1

`오류 메시지가 출력되는 상황`

왼쪽 괄호가 열린다음 닫히지 않은 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

a := (1 + 1)
ID: 1; CONST: 2; OP: 1;
(Warning) Missing right parenthesis ==> assuming right parenthesis at the end of statement

Result ==> a: 2;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - EOF가 인식된 시점에서)가 빠진것이 인식된다. 따라서 next_token은 다시)로 바꼈다가 EOF로 바뀐다.

a
:=

(
1
+
1
EOF
)
EOF

warning case 7-2 프로그램의 끝이 ;인 경우

`출력되는 오류 메시지`

`(Warning) There is semicolon at the end of the program ==> ignoring semicolon`

`예시 입력`

b := (1 + 1);

`오류 메시지가 출력되는 상황`

program의 끝이 ;인 경우

`위의 예시 입력에 대한 -v옵션이 없을 때의 예시 출력`

b := (1 + 1)
ID: 1; CONST: 2; OP: 1;
(Warning) There is semicolon at the end of the program ==> ignoring semicolon

Result ==> b: 2;

`위의 예시 입력에 대한 -v옵션이 있을 때의 예시 출력` - ;가 출력되지 않은 것을 볼 수 있다.

b
:=
(
1
+
1
)
EOF