프로그래밍 언어론

Report #4

2017.04.12

20154073

강다현

Q.. C언어의 한 구조를 선택하고, 이 구조에 대한 구문을 BNF와 EBNF로 각각 작성하라.

while문 선택 (while <condition> do <statement>)

BNF

<assign> 🡪 <statement-list>

<var> 🡪 A | B | C

<condition> 🡪 <var> == <var> | <var> != <var> |

<var> < <var> | <var> <= <var> //while문의 조건 비교

<statement> 🡪 <var > = <expression> |

{<statement-list>} |

if <condition> then <statement> [ else <statement> ] |

**while < condition> then <statement>**

//while 문 안에 while문이 중첩 가능, 조건문 사용 가능

<statement-list> 🡪 <statement> | <statement-list>; <statement>

<expression> 🡪 <factor> //수식들에 우선순위 배정

<factor> 🡪 <factor> \*\* <term> | <term>

<term> 🡪 <term> \* <expr> | <term> / <expr> |

<term> % <expr> | <expr>

<expr> 🡪 <expr> + <exp> | <expr> - <exp> | <exp>

<exp> 🡪 (<factor>) | <var>

EBNF

<assign> 🡪 <statement-list>

<var> 🡪 A | B | C

<condition> 🡪 <var> (== | != | < | <=) <var>

<statement> 🡪 <var > = <expression> |

{<statement-list>} |

if <condition> then <statement> [ else <statement> ] |

**while < condition> then <statement>**

<statement-list> 🡪 <statement>[;] { <statement-list> }

<expression> 🡪 <factor>

<factor> 🡪 <term> { \*\* <term> }

<term> 🡪 <expr> { ( \* | / | % ) } <expr>

<expr> 🡪 <exp> { (+ | -) <exp> }

<exp> 🡪 ( <factor> { ( <factor> ) } )

3장 연습문제

Q. 13. 문자 a가 n번 나오고, 다음에 문자 b가 n번 나오는 스트링들로 구성된 언어에 대한 문법을 작성하라. 여기서 n>0이다. 예를 들면, 스트링 abb, aabbbb, aaaabbbbbbbb 등은 언어에 속하지만, a, aabb, ba, aaabb 등은 속하지 않는다.

<S> 🡪 <AB> <A><B>

<AB> 🡪 a<B>b a | a<A>

<B> 🡪 <AB>b | b bb | bb<B>

Q. 19. BNF가 3.4.5절의 예제 3.6에서 제시된 것이고, 그 언어 규칙이 다음과 같은 속성 문법을 작성하라 : 데이터 타입이 식에서 혼합될 수 없으나, 배정문에서는 배정 연산자의 양변의 타입이 같을 필요가 없다.

1. 구문 규칙 : <assign> 🡪 <var> = <expr> //할당

의미론 규칙 : <expr>.expected\_type 🡨 <var>.actual\_type

**술어 함수 : <var>.actual\_type ( == | != ) <expr>.expected\_type**

1. 구문 규칙 : <expr> 🡪 <var>[2] + <var>[3] //식에서 데이터 타입 혼합 X

의미론 규칙 : <expr>.actual\_type 🡨

if(<var>[2].actual\_type = int) and

(<var>[3].actual\_type = int)

then int

**else if(<var>[2].actual\_type = real) and**

**(<var>[3].actual\_type = real)**

then real

end if

술어 함수 : <expr>.actual\_type == <expr>.expected\_type &

**<var>.[2].actual\_type == <var>.[3]actual\_type**

1. 구문 규칙 : <expr> 🡪 <var>

의미론 규칙 : <expr>.actual\_type 🡨 <var>.actual\_type

술어 함수 : <expr>.actual\_type == <expr>.expected\_type

1. 구문 규칙 : <var> 🡪 A | B | C

의미론 규칙 : <var>.actual\_type 🡨 look-up(<var>.string)

위의 구문 규칙에서 변경된 점은 1의 술어함수와 2의 의미론 규칙, 술어함수입니다. 우선 1에서는 배정문이 존재하고, 이 배정문의 의미는 의미론 규칙에서도 알 수 있듯이 <var>의 actual\_type을 <expr>의 expected\_type 할당하는 것과 같습니다. 하지만 문제에서 나와있듯이 양변의 타입이 같을 필요가 없으므로 술어함수를 추가하여 <var>의 actual\_type과 <expr>의 expected\_type이 같거나 다를 수 있다는 것을 명시해두었습니다.

다음으로 2에서는 식이 존재하는데, 우선 식부터 따지면 <var>[2]와 <var>[3]을 더하는 식이 있습니다. 하지만 문제에서 나와있듯이 데이터 타입이 식에서 혼합될 수 없다는 점이 명시되어 있습니다. 따라서 <var>[2]의 actual\_type과 <var>[3]의 actual\_type이 동시에 실수인 조건문을 추가하여야 합니다. 이 조건 외에 두 값의 actual\_type이 다른 조건은 데이터 타입이 식에서 혼합될 수 없다는 조건에 반하므로 배제합니다. 그리고 술어함수를 추가하여 두 타입이 같아야 함을 명시합니다.

느낀점

오늘 과제는 굉장히 흥미로웠고, 설계하는 과정이 꽤나 재밌었습니다. 규칙을 만들고 유도 하여 결과를 도출하는 것이 가장 좋았던 것 같습니다. 19번 문제를 이해하는데 조금 힘들었지만 그 외에 작년에 데이터 구조 시간에 사용했던 스파크 언어와 비슷한 느낌이 들었습니다.