**프로그래밍언어 개념 1주차 과제 보고서**

} else if (token == 'f') {

// false

// TODO: [Fill in your code here] match('f'); result = (boolean) false;

}

이 부분은 'f' 토큰, 즉 "false" 값을 처리하는 부분입니다.

'f' 토큰이 발견되면 match('f')를 호출하여 토큰을 소비하고, 결과를 (boolean) false로 설정합니다. 이로써 false 논리 값을 expr 메서드에서 처리하게 됩니다.

|  |
| --- |
| while (token == '&' || token == '|') { if (token == '&') {  // TODO: [Fill in your code here] match('&');  Object right = bexp(); result = (boolean) result && (boolean) right;  } else if (token == '|') {  // TODO: [Fill in your code here] match('|');  Object right = bexp(); result = (boolean) result || (boolean) right;  }  }  • |

이 부분은 '&'와 '|' 논리 연산자를 처리하는 부분입니다.

논리 연산자가 발견되면 해당 연산자를 소비하고, 오른쪽 피연산자를 **bexp** 메서드를 호출하여 해석합니 다.

그런 다음, **result**와 **right**의 논리 연산을 수행하고 결과를 **result**에 다시 저장합니다.

이로써 '&'와 '|' 논리 연산자를 처리하여 논리 표현식을 해석합니다.

// TODO: [Fill in your code here]

String operator = relop(); // 관계 연산자 분석 int aexp2 = aexp(); // 두 번째 산술 표현식 해석

// 연산자에 따라 비교하고 결과를 result에 저장

|  |
| --- |
| if (operator.equals("<")) { result = aexp1 < aexp2;  } else if (operator.equals("<=")) { result = aexp1 <= aexp2; } else if (operator.equals(">")) { result = aexp1 > aexp2;  } else if (operator.equals(">=")) { result = aexp1 >= aexp2; } else if (operator.equals("==")) { result = aexp1 == aexp2; } else if (operator.equals("!=")) { result = aexp1 != aexp2;  } else {  // 잘못된 연산자 처리 throw new RuntimeException("잘못된 관계 연산자：" + operator); } |

**String operator = relop();**:

**relop** 메서드를 호출하여 관계 연산자를 분석하고 해당 연산자를 **operator** 문자열에 저장합니 다.

**relop** 메서드에서는 다음과 같이 **<**, **<=**, **>**, **>=**, **==**, **!=** 중 하나의 연산자를 인식합니다.

**int aexp2 = aexp();**:

**aexp** 메서드를 호출하여 두 번째 산술 표현식을 해석하고 그 값을 **aexp2**에 저장합니다.

이러한 표현식은 관계 연산자의 오른쪽 피연산자 역할을 합니다.

연산자에 따라 비교: **operator** 문자열에 저장된 연산자에 따라 비교 연산을 수행하고, 결과를 **result** 변수에 저장합 니다.

**<** 연산자의 경우 **aexp1 < aexp2**로 비교하고, **<=**, **>**, **>=**, **==**, **!=** 연산자에 따라서도 각각 비교 를 수행합니다.

오류 처리:

만약 **operator**에 정의되지 않은 연산자가 들어오면, 예외를 발생시키고 해당 연산자가 잘못되 었음을 나타내는 메시지를 표시합니다.

|  |
| --- |
| // TODO: [Fill in your code here] if (token == '<') { match('<'); if (token == '=') { match('='); result = "<=";  } else { result = "<"; |
| }  } else if (token == '>') { match('>'); if (token == '=') { match('='); result = ">=";  } else {  result = ">";  }  } else if (token == '=') { match('='); if (token == '=') { match('='); result = "==";  } else {  result = "=";  }  } else if (token == '!') { match('!'); if (token == '=') { match('='); result = "!=";  } else {  result = "!";  } }  1. |

**if (token == '<')**:

입력으로 들어온 **token**이 **'<'**인 경우, 작동합니다. 이 경우, 다음 **match('<')** 호출로 **'<'** 토큰을 소비하고, 다음 토큰이 **'='**인지 확인합니다.

**'='** 토큰이면 **" <="** 문자열을 **result** 변수에 저장하고, 아니면 **"<"** 문자열을 저장합니다.

**else if (token == '>')**:

입력으로 들어온 **token**이 **'>'**인 경우, 작동합니다. 이 경우, 다음 **match('>')** 호출로 **'>'** 토큰을 소비하고, 다음 토큰이 **'='**인지 확인합니다.

**'='** 토큰이면 **">="** 문자열을 **result** 변수에 저장하고, 아니면 **" >"** 문자열을 저장합니다.

**else if (token == '=')**:

입력으로 들어온 **token**이 **'='**인 경우, 작동합니다. 이 경우, 다음 **match('=')** 호출로 **'='** 토큰을 소비하고, 다음 토큰이 다시 **'='**인지 확인합니다.

**'='** 토큰이면 **"=="** 문자열을 **result** 변수에 저장하고, 아니면 **"="** 문자열을 저장합니다.

**else if (token == '!')**:

입력으로 들어온 **token**이 **'!'**인 경우, 작동합니다. 이 경우, 다음 **match('!')** 호출로 **'!'** 토큰을 소비하고, 다음 토큰이 **'='**인지 확인합니다.

**'='** 토큰이면 **"!="** 문자열을 **result** 변수에 저장하고, 아니면 **"!"** 문자열을 저장합니다.

|  |
| --- |
| // TODO: [Modify code of aexp() for <aexp> -> <term> { + <term> | -  <term> }] int aexp() {  /\* expr -> term { '+' term } \*/ int result = term(); while (token == '+' || token == '-') { if (token == '+') { match('+'); result += term(); } else if (token == '-') { match('-'); result -= term();  } } return result;  }    // TODO: [Modify code of term() for <term> -> <factor> { \* <factor> | /  // <factor>}] int term() {  /\* term -> factor { '\*' factor } \*/ int result = factor(); while (token == '\*' || token == '/') { if (token == '\*') { match('\*');  result \*= factor(); } else if (token == '/') { match('/');  int divisor = factor(); if (divisor != 0) { result /= divisor;  } else {  // 0으로 나누기 오류 throw new ArithmeticException("제로 오류로 나누기");  }  } } return result;  } |

**aexp()** 메서드는 산술 표현식을 계산하는데 사용됩니다.

**result** 변수에 먼저 **term()** 메서드의 결과를 저장합니다. 그런 다음 **token**이 **'+'** 또는 **'-'**인 경우에만 반복적으로 루프를 실행합니다.

만약 **token**이 **'+'**인 경우, **match('+')**를 호출하여 **'+'** 토큰을 소비하고, 다음 **term()**의 결과를 **result**에 더합니다.

만약 **token**이 **'-'**인 경우, **match('-')**를 호출하여 **'-'** 토큰을 소비하고, 다음 **term()**의 결과를 **result**에서 뺍니다. 이러한 루프를 통해 모든 덧셈과 뺄셈 연산을 순차적으로 수행합니다. 최종적으로 **result**에는 산술 표현식의 결과가 저장되어 반환됩니다. **aexp()** 메서드는 산술 표현식을 계산하는데 사용됩니다. **result** 변수에 먼저 **term()** 메서드의 결과를 저장합니다. 그런 다음 **token**이 **'+'** 또는 **'-'**인 경우에만 반복적으로 루프를 실행합니다.

만약 **token**이 **'+'**인 경우, match**('+')**를 호출하여 **'+'** 토큰을 소비하고, 다음 **term**()의 결과를 result에 더합니다.

만약 **token**이 **'-'**인 경우, **match('-')**를 호출하여 **'-'** 토큰을 소비하고, 다음 **term**()의 결과를 **result**에서 뺍니다. 이러한 루프를 통해 모든 덧셈과 뺄셈 연산을 순차적으로 수행합니다. 최종적으로 **result**에는 산술 표현식의 결과가 저장되어 반환됩니다.   
  
**결과 화면** :  
文本

描述已自动生成