**Principles of Programming Languages**

Programming Assignment #2

20154098 염승윤

Class 구성

Main class

설명: 프로그램 시작점으로 파일 이름이 인자로 입력되면 파일을 읽어 LexicalAnalyzer 객체로 파일 내용을 넘겨줍니다.

LexicalAnalyzer class

설명: 정해진 문법에 따르는 지 판단하는 Parser를 Recursive Descent Parsing 기법을 이용하여 작성했습니다. Dynamic Scoped Language 이며 nonlocal변수의 참조는 Deep Access 방식으로 접근합니다.

변수

Int **next\_token**: 토큰을 정수형으로 저장합니다.

String **token\_string**: 토큰을 문자열 변수로 저장합니다.

HashMap<String, String> **simbolTable**: 변수 이름과 변수가 가지고 있는 값을 저장합니다.

String **code**: Main class에서 보낸 코드를 임시 저장합니다.

ArrayList<String> **ariStack**: 실행된 함수들의 AR 구조를 저장하는 스택 입니다.

ArrayList<String> **printStack**: print\_ari, 변수가 실행되었을 때 출력할 문자열을 저장하는

함수입니다. 실행중에 Syntax Error가 나면 출력되지 않습니다.

메소드

**lexical**: 입력 스트림을 분석하여 lexeme을 찾고 분석결과를 next\_token에 저장합니다.

**START**: 선언 되어있는 함수들 먼저 파싱하여 simbolTable에 선언되어있는 function 이름을

저장한다.

**FUNCTIONS**: main 함수를 찾아서 파싱 시작

**FUNCTION\_BODY**: 함수를 VAR\_DEFINITIONS로 넘겨줍니다. VAR\_DEFINITIONS 종료 후

ARI에 있는 정보를 삭제한다.(함수가 종료되었으니 stack에 쌓인 정보 제거)

**VAR\_DEFINITIONS**: variable 키워드로 시작하면 VAR\_DEFINITION을 변수, call, print\_ari

토큰이 나오면 STATEMENTS을 실행합니다.

**VAR\_DEFINITION**: variable 키워드 뒤에 있는 함수들을 AR(Activation Record)에 쌓고 만약

기존에 선언된 함수 이름 또는 지역 변수 이름과 동일하다면 Duplicate

declaration 에러 메시지를 출력하고 이 프로그램을 강제로 종료합니다.

**VAR\_LIST**: variable a, b 처럼 변수가 여러개 선언되어 있는 경우 콤마가 끝날때 까지

반복해서 변수를 AR에 쌓아 올립니다.

**STATEMENTS**: call, print\_ari, 변수 토큰이 나오면 해당 코드를 STATEMENT 함수로 전달하고

남아 있는 코드는 STATEMENTS 함수로 전달합니다.

**STATEMENT**: 1. call 키워드가 있으면 call 뒤에 문자열을 함수 이름으로 인식하고 해당

함수를 simbolTable에서 찾아 실행시킵니다.(해당 함수를 FUNCTION\_BODY

함수로 넘깁니다.)

2. prin\_ari 키워드가 있으면 ARI stack에 쌓여 있는 정보들을 출력합니다.

3. 변수 토큰이 있으면 해당 변수의 link\_count(동적링크 따라간 횟수)와

local\_offset(AR 구조 내에서 몇번째에 해당 변수가 저장되어있는지) 을

출력합니다.

**checkIdentName**: 변수 이름이 기존에 선언되어있는 함수 이름과 중복되는지, 변수 이름이

해당 함수 내에서 중복되는지 확인해 중복되면 에러 메시지를 띄우고

프로그램 종료 합니다.(함수 이름은 simbolTable, 변수 이름은 ARI stack 에서

확인합니다.)

**checkFunctionName**: 함수 이름이 중복되는지 확인해 중복되면 에러 메시지를 띄우고

프로그램 종료합니다.(함수 이름은 simbolTable에서 확인합니다.)

**printARI**: ARI stack 에 있는 정보(출력할 정보)를 printStack에 저장합니다.

**printIdent**: ARI stack에 있는 해당 변수의 link count 값과 local offset 값을 출력합니다.(link

count는 동적링크를 따라간 횟수, local offset은 AR 구조 내에서 몇 번째에

해당 변수가 저장되어있는지를 알 수 있습니다.)

**findFunction**: call 키워드가 불렸을 때 해당 함수 코드를 반환해줍니다.

**getCode**: 해당 함수의 코드를 가져옵니다. 함수 코드의 인덱스 값이 simbolTable에 저장되어있습니다.

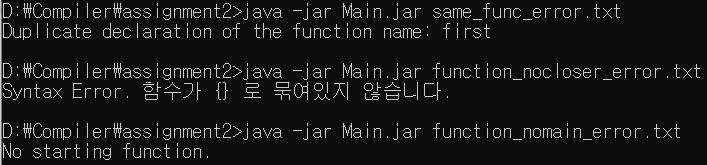
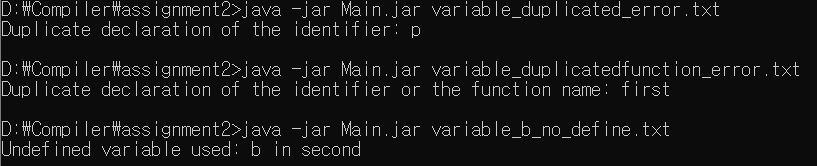
**getDynamicLink**: 해당 변수의 Dynamic link(동적링크 따라간 횟수)를 반환해줍니다.

**cleanAriStack**: 함수가 종료되었을때 그 함수에 해당하는 AR 정보를 제거합니다.

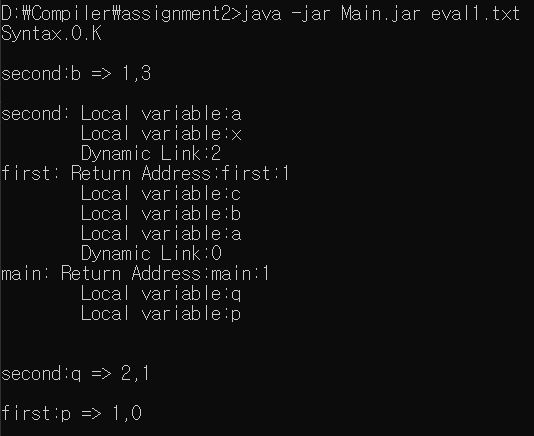
Error 메시지 나오는 경우

1. 같은 이름의 함수가 선언되었을 때 Duplicate declaration of the function name : <겹치는 함수 이름> 메시지가 출력됩니다.
2. 함수가 중괄호로 닫혀있지 않았을 때 ({ } 둘중 하나가 없을 때) 에러 메시지를 출력합니다.
3. main 함수가 없으면 No starting function 메시지가 출력됩니다.
4. 함수 내에서 같은 이름을 가진 지역 변수가 선언되면 Duplicate declaration of the identifier: <겹치는 변수 이름> 메시지를 출력합니다.
5. 변수 이름이 이미 선언된 함수 이름과 같다면 Duplicate declaration of the identifier or the function name: <겹치는 변수 이름>또는<겹치는 함수 이름> 둘중 하나만 메시지에 출력됩니다.
6. 선언되지 않은 변수를 부르면 undefined variable used: <선언 안된 변수 이름> in <에러가 발생한 함수 이름> 메시지가 출력됩니다.
7. 선언되지 않은 함수를 call 하면 Call to undefined function: <선언 안된 함수 이름> 메시지가 출력됩니다.

Error 메시지 실행 화면

1. same\_func\_error.txt
2. function\_nocloser\_error.txt
3. function\_nomain\_error.txt
4. variable\_duplicated\_error.txt
5. variable\_duplicatedfunction\_error.txt
6. variable\_b\_no\_define.txt
7. function\_undefinedcall\_error.txt

실행 화면

1. eval1.txt

2. eval2.txt

