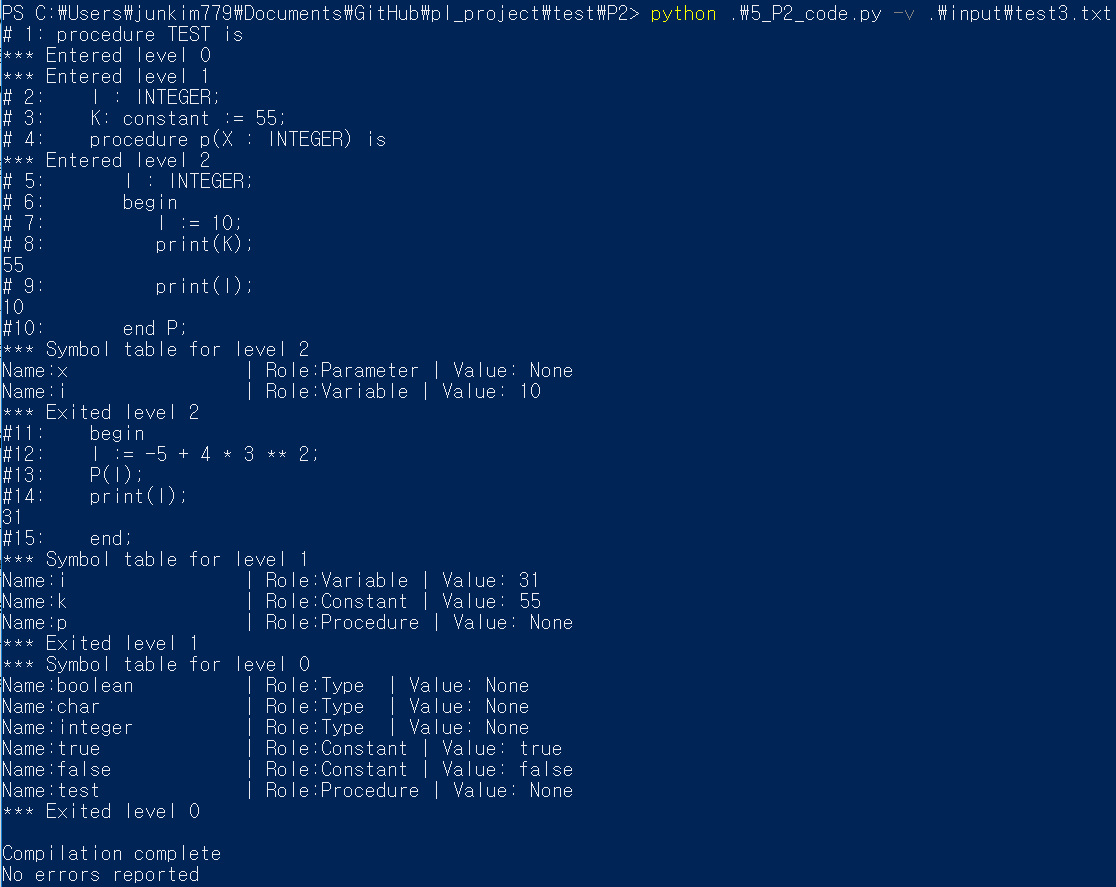
|  |  |
| --- | --- |
| 텀프로젝트 P2 설명서  CSI3103-01 프로그래밍언어구조론 | Python을 이용한 Static Semantic Analyzer 구현  5조: 김재민, 김준한, 마준영  2020년 11월 4일 |

텀프로젝트 P2 설명서

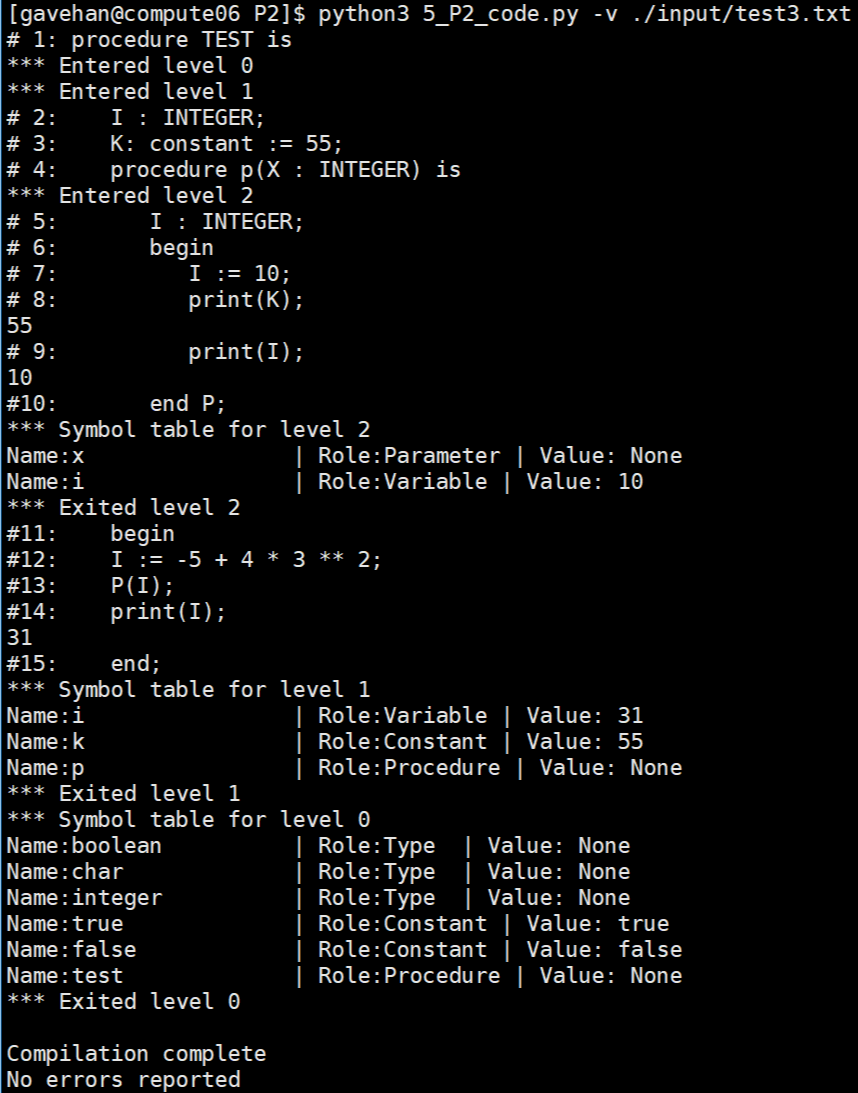
# 사용법

1. Windows
   1. Windows PowerShell를 실행한다.
   2. Windows PowerShell에서 cd {프로젝트 소스코드 경로} 명령어를 실행한다.
   3. Windows PowerShell에서 python ./5\_P2\_code.py {테스트 파일 경로} 명령어를 실행한다.  
      (경우에 따라, python이 아니라 python3일 수도 있다.)



* 1. 자세한 CUI 사용방법은 현재 문서 “모듈 소개”에서 확인할 수 있다.

1. Unix 계열
   1. cd {프로젝트 소스코드 경로} 명령어를 실행한다.
   2. python ./5\_P2\_code.py {테스트 파일 경로} 명령어를 실행한다.  
      (경우에 따라, python이 아니라 python3일 수도 있다.)



* 1. 자세한 CUI 사용방법은 현재 문서 “모듈 소개”에서 확인할 수 있다.

# Static Semantic Analysis 구현

1. SymbolEntry 클래스

Identifier의 semantic 특성들을 저장하고 관리하기 위한 클래스이다.

해당 symbol의 이름, 역할, 값 등을 저장한다.

1. SymbolTable 클래스

Statically managed symbol table의 구현이다. 개별 level의 symbol table은 identifier name을 열쇠로, SymbolEntry 인스턴스를 값으로 갖는 사전 자료구조로 표현한다. Scope analysis를 위해, SymbolTable 인스턴스는 사전으로 표현되는 symbol table을 스택 자료구조 속성을 통해 관리한다. 추가적으로, SymbolTable 클래스는 현재 탐색중인 block의 level을 나타내는 level 정수형 속성과 scope analysis 중 발생할 수 있는 오류를 전달할 Chario 인스턴스를 속성으로 갖는다.

1. Parser 모듈과 통합

Scope analysis와 role analysis를 위해서, 세 메서드가 추가되었다. Symbol entry가 기대하는 role인지 확인하는 accept\_role, symbol table에 신규 symbol entry를 추가하는 enter\_symbol, symbol table에서 주어진 identifier name에 해당하는 symbol entry를 찾는 find\_symbol이 추가되었다.

TinyAda의 경우, procedure가 scope block에 해당한다. 따라서, scope analysis를 위해서 procedure가 scope block에 해당하기 때문에, procedure specification 부분에서 새로운 scope level로 진입하고, procedure의 subprogram body 부분의 끝에서 scope level에서 이탈한다.

TinyAda의 경우, procedure 내부에 declarative 부분과 statements 부분이 분리되어 있고, statement 사용될 identifier들은 상위 block level, procedure의 매개변수, declarative 부분에서 선언된 identifier들로 구성된다. 따라서, role analysis를 위해서, 개별 role의 declaration에 해당하는 identifier를 symbol table에 추가하고, 해당 role을 설정한다. 그 뒤에, statements 부분에서 마주치는 identifier들은 symbol table에서 해당 identifier name의 symbol entry를 찾아보고, 그 symbol entry를 기대하는 role인지 확인한다.

# printStatement 구현

1. printStatement phrase를 추가한 EBNF
2. 수정된 EBNF 문법의 구현 방식
   1. statement

현재 토큰 인스턴스의 token\_id 속성이 “print” 예약어와 일치하는지 확인하는 부분을 조건문에 추가했다.

* 1. printStatement

accept\_token 메서드를 통해 “print”, “(“ 토큰을 확인한다. Expression phrase는 expression 메서드를 호출하고, expression에 해당하는 값을 문자열로 정상적으로 반환 받으면 출력한다. 그 후에, accept\_token 메서드를 통해 “)”, “;“ 토큰을 확인한다.

# Expression 정수 연산 구현

1. 연산 값의 EBNF phrase 메서드 간 전달

Expression 하위 phrase에 해당하는 메서드를 호출하면 문자열을 반환하도록 했다. 각 phrase 메서드의 문자열 반환값은 메서드 내부에서 실행된 정수 연산의 결과값에 해당한다. 상위 phrase 메서드에서는 하위 phrase에 해당하는 메서드를 호출하고 그 반환값을 지역 변수로 저장하고, 이를 이용하여 추가 연산 및 확인을 실행한다. 추가적으로, 정수 연산이 아니거나, 지원하지 않는 연산의 경우에는 (예를 들어, 불리언 자료형이 반환되는 expression) None을 반환하여, 상위 phrase 메서드에서 확인을 용이하게 했다.

1. check\_number 유틸리티 함수

Phrase 메서드의 문자열 반환값이 유효한 정수값인지 확인하는 유틸리티 함수이다. 문자열의 첫번째 문자가 “-“인지 확인하고, “-“ 존재 여부에 맞추어 문자열이 숫자로만 구성되어 있는지 확인한다. 확인 결과는 불 자료형을 반환하여 전달한다.

1. 정수 연산

하위 phrase의 값(들)을 반환 받은 뒤, 해당 반환값(들)을 check\_number 함수를 통해 유효한 정수인지 확인한 후에, 유효한 정수 값(들)이라면 정수로 자료형 변환한 값(들)에 대해 연산(들)을 진행한다. 더 상세한 설명은 “모듈 소개”에서 확인할 수 있다.

# 모듈 소개

1. **CUI (5\_P2\_code.py)**
   1. Logger 클래스

stdout(터미널 화면)과 주어진 파일 경로의 로그 파일에 동시 출력 기능을 제공한다.

* 1. receive\_args 함수

커맨드라인에서 옵션을 받는다. 옵션은 “-v”, “-o”가 있고, 필수 입력 항목은 TinyAda 소스 프로그램 파일 경로이다. “-v” 옵션을 준 경우, 입력 소스 프로그램 전체가 출력되며, 필수로 오류 보고를 출력한다. “-o” 옵션을 준 경우, syntax analyzer의 출력을 stdout(터미널 화면)과 주어진 파일 경로의 로그 파일에 동시에 출력한다.

* 1. main

커맨드라인에서 받은 옵션들과 입력 소스 프로그램 파일로 Chario, Scanner, Parser 클래스들의 인스턴스를 각각 생성한다. 그 후에, Parser 클래스에서 parsing을 시작한다. Parsing 중, syntax 오류가 발생한 경우, 예외 처리를 통해서, 오류를 출력하고 종료한다. Parsing 중 오류가 syntax 오류가 발생하지 않은 경우에, lexical 오류가 있다면 보고하고, 만약 오류가 없었다면 출력이 없다.

1. **Token (token.py) – Token 클래스를 통해 parsing 중 여러 모듈 간 상호작용을 용이하게 한다.**

P1 구현과 달리, printStatement 구현을 위해, 예약어 집합에 “print”가 추가되었다. 또한, 유효한 토큰 종류에서 “type”을 제거했다. 그 이유는 해당 토큰 종류의 기능을 SymbolEntry와 SymbolTable이 대체하기 때문이다.

1. **Chario (chario.py) – TinyAda 소스 프로그램의 문자를 읽어 들이고, parsing 중 오류를 보관한다.**

P1 구현과 동일하다.

1. **Scanner (scanner.py) – Chario에서 문자 스트림을 전달받아, 토큰을 인식한다.**

P1 구현과 동일하다.

1. **Parser (parser7.py)**
   1. Static semantic analysis를 위한 자원 준비

symbol\_entry.SymbolEntry 클래스를 SymEnt, symbol\_table.SymbolTable 클래스를 SymTab로 import 한다. Symbol entry role analysis를 위해, L\_NAME\_SET과 R\_NAME\_SET를 도입한다.

* 1. printStatement를 위한 자원 준비

“print” 명령어를 수행하기 위해 STATEMENT\_OP\_SET에 “print”를 추가했다.

* 1. Parser 클래스 초기화 수정

Parser 클래스 table이라는 SymbolTable 오브젝트 속성을 추가했다. table 속성은 클래스 초기화 과정 중에서, enter\_scope 메서드를 호출하여, 기본 scope (level = 0)을 설정하고, enter\_symbol 메서드를 통해서, 기본 symbol들인 boolean, char, integer, true, false를 symbol table에 입력한다.

* 1. \_\_test\_token 메서드 이름을 \_\_accept\_token으로 변경
  2. \_\_accept\_role 메서드 추가

주어진 SymbolEntry instance의 role과 기대되는 role이 일치하는지 확인한다. 만약 role이 일치하지 않는다면, 전달받은 오류 메시지를 가지고 오류를 발생시킨다.

* 1. \_\_enter\_symbol 메서드 추가

현재 symbol table에 새로운 symbol을 추가한다. 문자열 형태의 role과 name을 전달받아 내부적으로 SymbolTable의 enter\_symbol 메서드를 호출한다. name이 주어지지 않을 경우, 현재 token이 identifier인지 확인하고, 해당 토큰의 정보를 사용하여 enter\_symbol 메서드를 호출한다. name이 주어지지 않고, 현재 token이 identifier가 아니라면, 예외를 발생시킨다. Symbol table에 새로 추가되는 identifier에 상응하는 SymbolEntry instance를 반환한다.

* 1. \_\_find\_symbol 메서드 추가

현재 symbol table에서 주어진 symbol을 찾는다. 문자열 형태의 name을 전달받아 내부적으로 SymbolTable의 find\_symbol 메서드를 호출한다. name이 주어지지 않을 경우, 현재 token이 identifier인지 확인하고, 해당 토큰의 정보를 사용하여 find\_symbol 메서드를 호출한다. name이 주어지지 않고, 현재 token이 identifier가 아니라면, 예외를 발생시킨다. Symbol table에서 찾은 identifier에 상응하는 SymbolEntry instance를 반환한다.

* 1. \_\_print\_statement 메서드 추가

Print 함수의 괄호 속, expression을 연산하고, 그 값을 출력한다.

* 1. compilation() 메서드 수정

Parsing 완료 후, 기본 scope (level 0)을 이탈하기 위해, exit\_scope() 메서드를 호출한다.

* 1. \_\_subprogram\_body() 메서드 수정

해당하는 block을 이탈하기 위해서, exit\_scope() 메서드를 호출한다. Subprogram body 마지막에 procedure name이 다시 언급되는 경우, accept\_role 메서드를 통해, 해당 identifier의 role이 “proc”인지 확인한다.

* 1. \_\_subprogram\_spec() 메서드 수정

현재 procedure name에 해당하는 identifier를 symbol table에 추가하고, role을 “proc”으로 설정한다. 신규 block에 진입하므로, enter\_scope() 메서드를 호출한다.

* 1. Declaration phrase 메서드들 수정
     1. Number declaration: 해당 identifier에 상응하는 symbol entry를 symbol table에 추가하고, role을 “const”로 설정한다. 또한, phrase의 expression의 반환값을 symbol entry의 값으로 설정한다.
     2. Object declaration: 해당 identifier에 상응하는 symbol entry를 symbol table에 추가하고, role을 “var”로 설정한다.
     3. Type declaration: 해당 identifier에 상응하는 symbol entry를 symbol table에 추가하고, role을 “type”로 설정한다. 기존의 type을 이용하여 선언하는 type의 경우, 해당 type에 상응하는 identifier가 symbol table에 존재하는지 확인하고, role이 “type”인지 확인한다. 추가적으로, enum declaration의 경우에, 해당 enumeration 값(들)의 identifier에 상응하는 symbol entry를 symbol table에 추가하고, role을 “const”로 설정한다.
  2. \_\_identifier\_list() 메서드 수정

인식된 identifier(들)에 상응하는 SymbolEntry 인스턴스를 반환한다.

* 1. \_\_parameter\_specification() 메서드 수정

현재 parameter name(s)에 해당하는 identifier(들)을 symbol table에 추가하고, role을 “param”으로 설정한다. 또한, 각 매개변수의 type에 해당하는 identifier(들)이 symbol table에 존재하고, 이(들)의 role이 “type”인지 확인한다.

* 1. \_\_assign\_call\_statement() 메서드 수정
     1. Variable assignment 경우, 현재 identifier에 상응하는 symbol entry를 symbol table에서 찾고, role이 “var”이나 “param”인지 확인한다. Role이 유효한 경우, 이어지는 expression에 해당하는 값을 symbol entry의 val 속성에 저장한다.
     2. Procedure call 경우, 현재 identifier에 상응하는 symbol entry를 symbol table에서 찾고, role이 “proc”인지 확인한다.
  2. Expression과 예하 phrase 메서드들 수정
     1. expression: 해당 expression의 정수 연산값을 문자열로 반환한다. 논리 연산은 지원하지 않으므로, 논리 연산자가 포함된 경우, None을 반환한다.
     2. relation: 논리 연산은 지원하지 않으므로, 논리 연산자가 포함된 경우, None을 반환한다.
     3. simple\_expression: 산술 연산 중 마지막 순위인 덧셈, 뺄셈을 수행한다. 우선 unary sign operator에 대해 평가한다. 피연산자가 (term 메서드 반환값) 유효하면 부호를 설정한다. 그 후에 연산자를 파악하고, 이전 연산 결과값과 신규 피연산자의 (term 메서드의 반환값) 유효성을 확인하고, 유효하면 연산을 한 뒤, 결과값을 기존의 연산 결과값 변수에 저장한다. 여러 연산자가 연속되면, 반복문을 통해서 연산을 수행한다. 연산 결과값이 유효한 정수이면, 그 값에 해당하는 문자열을 반환하고, 유효하지 않으면, None을 반환한다.
     4. term: 산술 연산 중 중간 순위에 있는 곱셈, 나눗셈, 나머지 연산을 수행한다. 연산자를 파악하고, 이전 연산 결과값과 (또는, 최초 factor 메서드 반환값) 신규 피연산자의 (factor 메서드의 반환값) 유효성을 확인하고, 유효하면 연산을 한 뒤, 결과값을 기존의 연산 결과값 변수에 저장한다. 여러 연산자가 연속되면, 반복문을 통해서 연산을 수행한다. 연산 결과값이 유효한 정수이면, 그 값에 해당하는 문자열을 반환하고, 유효하지 않으면, None을 반환한다.
     5. factor: 산술 연산 중 우선 순위에 있는 제곱 연산을 수행한다. 연산자를 파악한다. 부정 연산자의 경우, 논리 연산은 지원하지 않으므로, None을 반환한다. 제곱 연산자인 경우, 두 피연산자(primary 메서드의 반환값들)을 연산한다. 연산 결과값이 유효한 정수이면, 그 값에 해당하는 문자열을 반환하고, 유효하지 않으면, None을 반환한다.
     6. primary:

현재 토큰이 괄호인 경우, 산술 연산 중 최우선 순위에 있는 괄호 내 연산을 수행한다. 괄호가 있는 경우 expression 메서드를 호출해서 연산 결과값을 반환한다.

현재 토큰이 identifier인 경우, 그에 상응하는 symbol entry를 symbol\_table에서 찾고, 그 role이 “var”, “param”, “const” 중 하나인지 확인한다. Role이 유효하면, symbol entry의 값을 반환한다.

현재 토큰이 literal 정수인 경우 그 값을 반환한다.

* + 1. name: 현재 토큰에 상응하는 identifier를 symbol table에서 찾아, SymbolEntry 인스턴스를 반환한다.

1. **SymbolEntry (symbol\_entry.py) – Identifier에 대한 semantic 정보를 저장한다.**
   1. SymbolEntry 클래스

Identifier에 대한 정보들을 기록하는 클래스이다. 속성으로 name, role, val, next를 가진다.

name은 문자열 자료형으로 identifier의 이름을, role은 문자열 자료형으로 identifier의 역할을, val은 문자열 자료형으로 identifier의 값을 저장한다. Identifier의 role로 유효한 값은 const, var, type, proc, param이다. const는 constant를, var는 variable을, type은 type을, proc는 procedure를, param은 parameter를 의미한다. next는 appended된 SymbolEntry instance를 가리킨다.

* 1. append 메서드

SymbolEntry instance를 받아서 가장 마지막의 next에 append한다.

* 1. set\_role 메서드

Identifier의 role을 문자열로 받아서 SymbolEntry instance의 role 속성에 저장한다. 이때, role은 위에서 설명한 5가지의 유효한 role에 속해야 한다.

* 1. set\_value 메서드

Identifier의 값을 문자열 형태로 받아서 SymbolEntry instance의 val 속성에 저장한다.

1. **SymbolTable (symbol\_table.py) – Block level에 따라, SymbolEntry들을 관리한다.**
   1. SymbolTable 클래스

각 block level에서 존재하는 identifier들을 identifier name을 열쇠로, SymbolEntry 인스턴스를 값으로 갖는 사전 자료구조로 구성된 stack 자료구조 형태로 보관하는 클래스이다. 속성으로 stack, level, chario를 가진다. stack은 사전을 값으로 하는 리스트 자료구조로, symbol table stack을 저장한다. level은 정수 자료형으로, 현재 parsing 중인 block의 level을 저장한다. chario는 static semantic analysis 중 발생하는 오류를 전달하기 위해 Chario 클래스 인스턴스를 저장한다.

* 1. enter\_scope 메서드

새로운 block level에 진입한 것을 표현한다. level 속성을 증가하고, 새로운 symbol table을 의미하는 사전 자료구조를 stack에 push한다.

* 1. exit\_scope 메서드

기존 block level에서 벗어난 것을 표현한다. level 속성을 감소하고, symbol table을 의미하는 사전 자료구조를 stack에서 pop한다. Chario 인스턴스의 is\_verbose 속성의 값에 따라, pop된 symbol table을 출력한다.

* 1. \_\_print\_table 메서드

하나의 symbol table을 의미하는 사전 자료구조의 내용을 출력한다.

* 1. enter\_symbol 메서드

현재 stack의 최상위 symbol table에 새로운 symbol entry를 입력한다. Symbol table에 새롭게 입력하려는 symbol entry에 해당하는 identifier name이 이미 있는 경우, chario를 통해서 오류를 전달한다.

* 1. find\_symbol 메서드

현재 stack의 최상위 symbol table에서 주어진 symbol entry를 검색한다. Symbol table에 주어진 symbol entry에 해당하는 identifier name이 없는 경우, chario를 통해서 오류를 전달한다.

1. **프로젝트의 파이썬 모듈에 있는 주석을 통하여 추가적인 설명, 사용 예시에 대해 확인할 수 있다.**

# 후기

**김재민 (담당 파트: SymbolTable 구현 | Token, Chario, Scanner 수정)**

이번 프로젝트에서는 syntax 뿐만 아니라 static semantic analyzer를 구현하면서 우리가 평소에 아무 생각 없이 사용하는 프로그래밍 언어가 어떠한 복잡성을 가지는지를 깨달을 수 있었다. 또한 단순한 분석 뿐 아니라 print와 기본적인 정수형 연산을 구현하면서 당연해보이는 연산 안에 얼마나 많은 생각이 얽혀 있는지를 몸소 체험해 볼 수 있었다.

**김준한 (담당 파트: Parser 통합 | Expression 정수 연산)**

이번 프로젝트를 진행하면서, semantics에 대해 많이 배울 수 있었다. 특히, 조원들과 함께 디버깅을 하면서, 오류라고 생각한 부분들이 scope 과 role이 아닌, data type과 관련된 사항들이란 것을 깨달았다. 이런 활발한 토의를 통해서, 이전 수업들에서는 되게 단순하게 받아들인 프로그래밍 언어의 semantic 특성들을 보다 명료하게 이해할 수 있어서 좋았다.

**마준영 (담당 파트: SymbolEntry 구현 | printStatement 구현)**

이번 프로젝트를 통해 static semantic analyzer에 대해 배울 수 있었다. Static semantic analyzer 이후 단계까지 진행하지는 않았지만, 프로젝트를 단계별로 진행하면서 프로그래밍 언어 구현이 진행되는 방식을 몸소 느낄 수 있었다. 팀원들과 함께 프로젝트를 진행하면서 협동심을 기를 수 있었다.