Design and Development of Compiler  
for C- Language  
(설계 프로젝트 수행 결과)

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 20131612 최대운

개발기간: 2019. 4. 16. ~ 2019. 5. 03.

각 단계별 결과 보 고 서

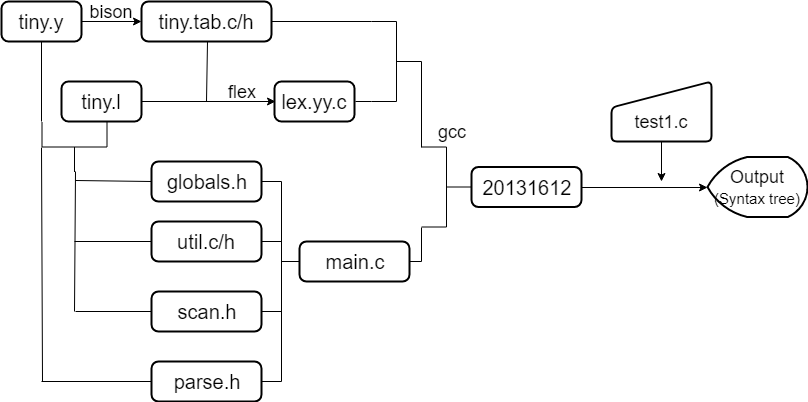
프로젝트 제목: Design and Development of Compiler for C-Language:  
 Phase 2: Design and Implementation of LALR Parser

제출일: 2019. 5. 03.  
개발자: 20131612 최대운

1. 개발 목표  
   - Bison과 프로젝트1에서 구현했던 Lexical analyzer를 이용해 C- parser를 구현한다.  
   - tiny.y 파일을 적절히 수정하여 C-용 BNF Grammar를 만든다. BNF Grammar는 교재 부록에 제시되어 있다.  
   - 입력파일에 대해 BNF Grammar를 기반으로 Syntax Tree가 만들어지면 적절한 방법으로 출력한다.
2. 개발 범위 및 내용
   1. 개발 범위  
      - 프로젝트1에서 만들었던 Lexical analyzer와 Bison을 이용하여 Syntax Tree 제작.
   2. 개발 내용  
      - C- 언어에 맞게 BNF Grammar 수정  
      - BNF Grammar에 따른 트리 노드 생성 및 연결
3. 추진 일정 및 개발 방법
   1. 추진 일정

|  |  |
| --- | --- |
| 일정 | TODO |
| 04. 30.~ 05. 02 | 프로젝트 명세서 파악 및 개발 |
| 05. 02. | 보고서 작성 |
| 05. 03. | 제출 |

* 1. 개발 방법  
     - 프로젝트1에 추가하여 Bison을 이용해 Syntax Tree 구현 및 출력

1. 연구 결과
   * 1. 합성 내용  
          
        <프로그램 구조도>

이번 프로젝트를 진행할 때, 주의해야 할 점은 프로젝트 1에서 globals.h에 정의했던 TokenType을 사용하지 않고 tiny.y에 token들을 정의해서 사용해야 한다. 다음 tiny.y를 bison을 이용해 Parser를 만들면 tiny.tab.c, tiny.tab.h 파일이 생성되고 tiny.tab.h를 tiny.l 파일 안에 포함시켜 token을 사용하게 된다. 이때, ENDFILE token은 정의되지 않기때문에 globals.h에 0값으로 define 해준다.

* + 1. 분석 내용  
       프로젝트 1에 이어서  
       - main.c 파일에선 메인 함수 내에서 tiny.y에 정의되어 있는 parse함수를 호출하여 Syntax Tree의 Root를 얻는다.

- globals.h 파일에선 프로젝트 1에서 사용하던 enum 형식의 TokenType을 int형으로 다시 선언하여 기존 프로젝트의 큰 변화가 없게 한다. 또, C- 언어에 존재하는 선언문을 구분하기 위한 enum 값 들을 추가한다.

- util.c/h 파일에서는 C-언어에서 선언문에 해당하는 노드를 만들기 위한 함수가 정의 되어야 하고, Syntax Tree가 출력되는 함수가 구현되게된다.

- scan.h와 parse.h는 getToken함수와 parse함수의 원형을 갖는다.

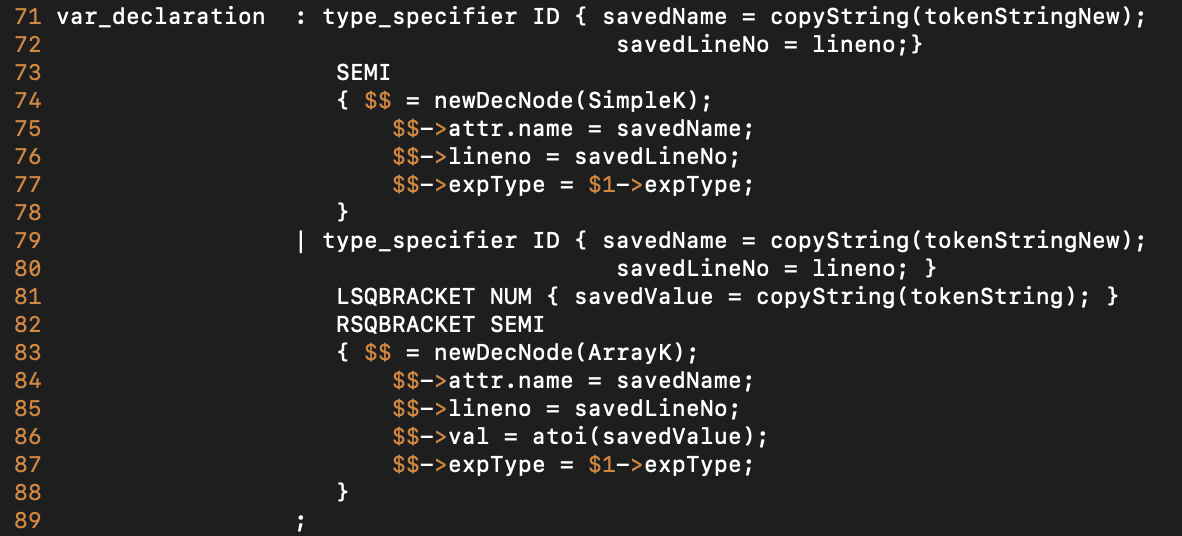
* + 1. 제작 내용

메인 함수에서는 스켈레톤 코드에서 TraceParse Flag를 True로 만들고 TraceScan Flag를 False로 만들어서 parse함수를 통해 얻어온 Syntax Tree Root를 printTree 함수의 파라미터로 넣어 Syntax Tree가 출력될 수 있도록한다.

globals.h에서는 enum 타입 NodeKind에 DeclarationK를 추가하여 변수나 함수선언부 노드를 생성할 수 있게 한다. 그리고 DeclarationKind와 관련한 SimpleK, ArrayK, FunctionK, ParamK를 enum 타입으로 선언한다. 이는 각각 일반 변수 선언, 변수 배열 선언, 함수 선언, 파라미터 선언으로 구분된다. 이에 따라 treeNode 구조체의 union kind에 DeclarationKind dec 변수를 추가한다.

tiny.l에서 tokenString과 같은 자료형, 같은 크기를 갖는 tokenStringNew를 선언하여 tokenString이 바뀌기 직전에 tokenStringNew에 tokenString을 복사한다. 이렇게 하는 이유는 LALR(1) parser에서 ID를 얻는 과정에서 다음 Lookahead를 볼 때 tokenString이 바뀌게 되어 ID에 해당하는 tokenString을 얻지 못하기 때문이다.

tiny.y에서 BNF grammar 작성은 교재의 부록 A.2를 따라 작성한다. 이 때, match되는 각 Rule에 대해 노드를 생성할 때 중간에 등장하는 token ID, NUM, 연산자들을 저장하기 위해 savedName, savedFuncName, savedValue, savedRelOp, savedAddOp, savedMulOp 변수를 선언하여 사용한다. 사용 예는 아래와 같다. ε을 표현하기 위해서는 %empty 로 rule을 표현할 수 있다. 교재에서 주어진 BNF grammar는 selection\_stmt에서 Shift-Reduce conflict가 발생한다. 본 과제에서는 우선 선택사항을 설정하지 않기 때문에 Shift를 우선 하게 된다.

  
<MidRule을 사용해 ID, NUM을 저장하는 예>

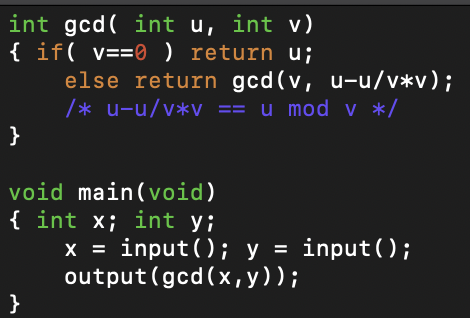
util.c에서는 newDecNode 함수를 추가한다. 이 함수는 DeclarationKind를 파라미터로 받고 선언부에 해당하는 노드를 생성하는 역할을 한다.  
printTree 함수의 전체적인 구조는 파라미터로 들어온 tree의 각 nodekind와 kind가 가질 수 있는 타입별로 적절한 출력을 한다. 다시 말하면, tree->nodekind와 tree->kind.\*별로 비교하게 된다. 적절한 출력이라 함은 statement의 속성에 이름이 들어가는 경우엔 이름까지 출력하고 Constant 값인 경우 그 값까지 출력하는 것을 말한다. 이후 tree의 child가 NULL이 아니라면 child를 printTree의 파라미터로 넣어 재귀호출하고 마지막으로 tree의 sibling으로 넘어가는 방식으로 짜여져있다. printTree 함수 시작과 끝에 INDENT와 UNINDENT가 있어 printTree 함수가 중첩 되어 호출 됨에 따라 tree의 depth를 표현하게 된다. 다만 예제에서 주어진 출력 형식에 따라 tree->nodekind가 DeclarationK 이고 tree->type.dec가 FunctionK, ParamK일 때, type을 출력해주기 위해 조건문 안에서 INDENT를 하고 type을 출력한 후 UNINDENT하는 방식으로 구성되어있다. 이 외의 경우에 대해선 ID 밑에 type이 같은 INDENT Line에 나온다. 또 ParamK의 경우 파라미터가 없을 땐, Parameter: (null)을 출력해준다. ArrayK의 경우 type은 Array로 출력되고 Array 사이즈가 추가로 출력된다.

텍스트이(가) 표시된 사진

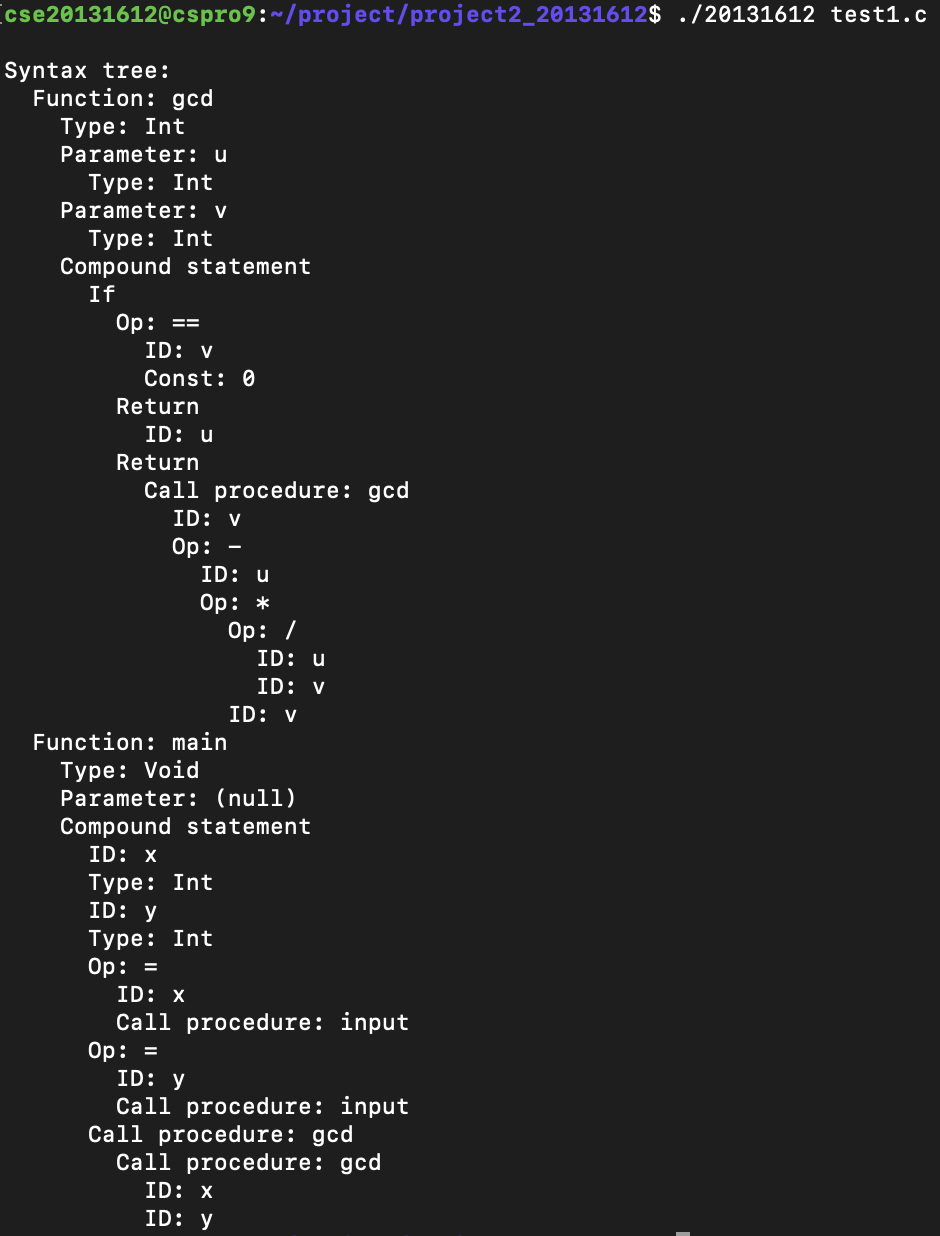
자동 생성된 설명  
<FunctionK인 노드의 출력 예>

* + 1. 시험 내용  
       제시된 입력 결과에 대한 결과는 아래와 같다.

<입력 파일의 예>



<입력파일에 대한 출력결과>



Grammar Rule와 연관하여 token에 에러가 발생했을 때는, 해당 라인과 token을 출력하고 parsing을 멈춘다.

1. 기타
   1. 자유 기술

Bison을 이용해 LALR(1) parser를 만들고 여기에 입력을 넣어 Syntax Tree를 만들어 볼 수 있었다. 궁금한 부분은 ID Token을 얻어올 때, ID Token이 끝난 다음의 Token이 넘어오는 경우가 있어서 tokenStringNew를 추가해야했다. 이게 lex의 문제인지 bison의 문제인지 알 수가 없었다.