Design and Development of Compiler

for C- Language

(설계 프로젝트 수행 결과)

과목 명 : [CSE4120] 기초 컴파일러 구성

담당 교수 : 정성원 교수님

개발자 : 20141602 황기덕

개발 기간 : 2019/04/27~2019/05/03

프로젝트 제목 : Design and Development of Compiler for C-Language:

Phase 2: Design and Implementation of LALR Parser

제출일 : 2019년 5월 03일

개발자 : 황기덕

1. 개발 목표

* 1차 프로젝트에서 구현한 lexical analysis를 바탕으로 C- parser를 만들고 abstract syntax tree를 구현, print tree로 그 결과를 보여준다.

1. 개발 범위 및 내용
   1. 개발 범위

* C- parser

cm.y 안에 rule 를 정의하고 LALR parser를 만들고 ast를 도출해낸다.

* Print tree

util.c 파일에서 print tree를 추가해 제작한 트리를 보여준다.

* 1. 개발 내용
* C- parser

각각의 토큰들이 어떤 우선순위를 가지고 있는지, assoc는 어떻게 되어 있는지 정해준다.

각각의 grammar에서 tree를 만들기 위해 어떤 행동을 해야 하는지 정의해준다

* Print tree

tree를 preorder처럼 탐색하면서 본인의 tree의 nodekind, kind를 보고 출력 양식을 결정해준다.

1. 추진 일정 및 개발 방법
   1. 추진 일정

04/27 : 요구사항 분석

04/28 ~ 05/02 : 프로젝트 개발

* 1. 개발 방법
* C- parser/ print tree

flex를 이용하여 token화 한 다음 그 토큰 된 것들을 yacc을 이용하여 abstract syntax tree를 만들어 준다. 그 후 만들어진 tree를 util.c 에서 형식에 맞게 출력해 준다. 그러기 위해 util.c, h 를 수정하고 global.h에 알맞은 변수를 추가해주며, cm.y에 declaration, translation rule를 추가해준다.

* 1. 추가 코드
* Makefile

실행 파일과 lex.yy.c 등의 파일을 만들고, 오브젝트 코드등을 만들고 컴파일을 하기 쉽게끔 추가해주었다.

* readme.txt

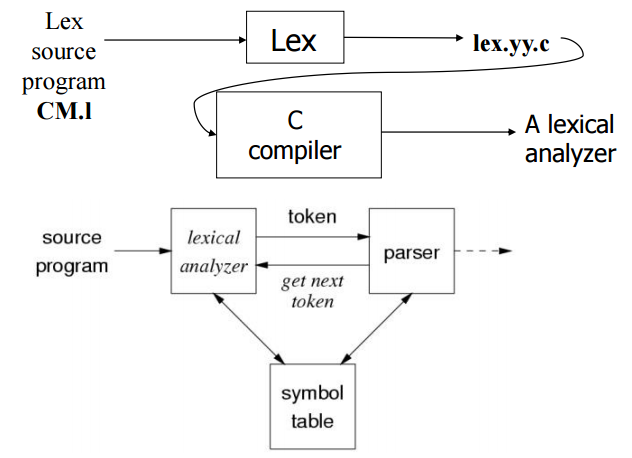
간단한 사용방법을 적어두었다.

* error.c / errorfree.c / gcd.c / minloc.c

평가를 위해 만들어 둔 c- 코드이다.

1. 연구 결과
   1. 합성 내용

* 전체 구조



1차 프로젝트에서 완성시킨 cm.l을 이용하여 flex를 통하여 token화한다. 그 후 parser에서 lexical analyzer를 통해 나오는 토큰을 계속 받으며 LALR parser를 실행시키며 syntax가 cm.y에서 정의해둔 문맥에 맞는지 검사하며 동시에 tree를 만든다. 에러가 있다면 도중에 종료하고 에러 메세지를 출력해 주며 라인 넘버를 보여준다.

* 1. 분석 내용

위와 같은 일을 하기 위하여 총 다섯 개의 파일을 수정하고, makefile을 그 의존성에 맞게 수정해 주었다.

* global.h

tree의 기본 node 형태가 저장되어 있고, 각각의 노드가 어떤 타입인지 선언할 수 있게끔 enum을 정의해 두었다. 1차때 lexical analysis를 위하여 정의해 둔 TokenType은 flex/yacc을 하는 동안 생성이 되므로 주석처리한다.

* scan.h

translation rule을 정할 때 token의 lexeme을 받아야 하는데, 그 token을 저장해 두어서 사용할 수 있게끔 한다.

* util.h

cm.y와 util.c 에서 트리 노드를 생성하는 데 필요한 함수를 정의해 주어야 한다.

* util.c

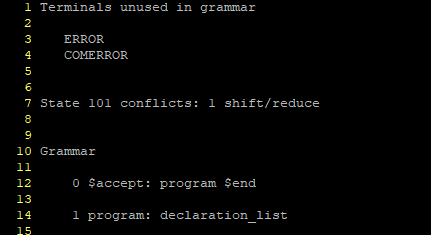
tree 생성 이후 print tree를 할 때 각 노드 타입이 들어왔을 때 어떠한 방식으로 출력해야 하는지 정의해 주고, 실제로 출력한다. 이미 만들어져있는 node type말고 새로운 dec type을 선언해 주고, 그 노드를 생성할 때 어떤 작업을 하는지 정의해 준다.

* cm.y

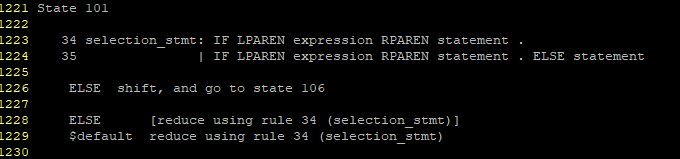
실제로 parser 역할을 수행하고, parsing 하면서 트리를 만들어줄 수 있게끔 각 토큰을 정의해주고, 그 토큰 별 우선순위와 left/right associative, shift reduce conflict가 일어날 때 어떻게 해야 하는지, C- 문법에 맞게 syntax를 정의하고 그에 따라 translation rule을 정의해 parsing을 하며 tree를 만들 수 있게끔 해 준다.

* 1. 제작 내용
* Handling conflict

shift-reduce conflict와 reduce-reduce conflict가 존재한다고 배웠다. 이 문법에서는 shift-reduce conflict가 발생한다고 나온다. yacc을 할 시 -v 옵션을 주어서 y.output을 확인해 보면



이러한 그림을 볼 수 있다.



shift reduce conflict가 났다는 위 그림을 보면 위와 같이 if ( expression ) stmt . ELSE 상황이 발생했을 때 selection statement로 reduce 해야하는지, 아니면 ELSE로 shift해야 하는지에 대해 conflict가 난다.

if(True)

if(False)

printf(“1”);

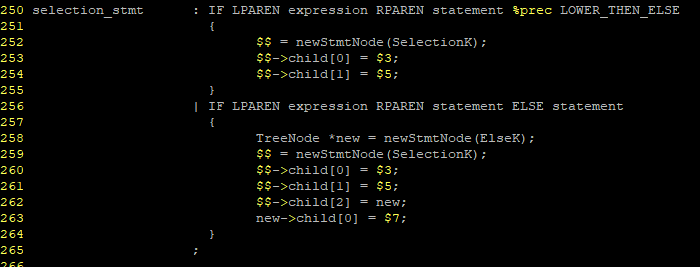
else

printf(“0”)

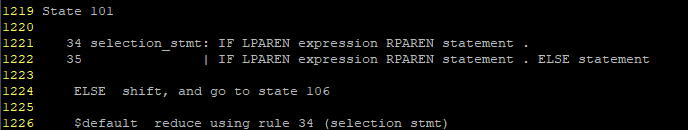
위를 실행하면 0이 나와야 한다. 즉 state 101과 같은 상황이 발생하면 shift action을 취해야 else가 알맞은 짝을 지을 수 있다. 그러므로 저 상황이 나오면 우선순위를 두어 shift를 우선적으로 하게 만들어 주어야 한다. 그러므로 cm.y에서



이 두 문장을 추가해주고



이렇게 해 주어서 shift/reduce conflict가 나면 shift를 우선적으로 할 수 있게 해결해 주었다.



* Error handling

syntax error가 날 시 현재 내가 어떠한 토큰을 받았고 그 토큰과 lexeme이 무엇인지, 이전에는 어떠한 토큰이 있었는지를 현재 line number와 함께 출력해 주었다.

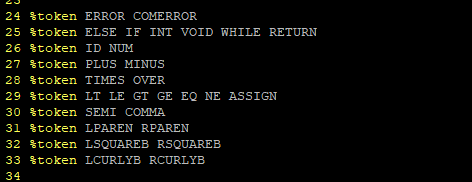
* run program

사용한 grammar rule은 교재 491~493 page에 나와있는 rule을 그대로 사용하였다.

cm.y에 grammar를 넣어 주었으며, 그 위에서 우선순위에 대한 정의를 해 주어서 어떤 순서로 갈지 정해주었다. 프로그램을 실행할 때 flex와 yacc을 해 주어서 토큰화를 하고 토큰을 바로 받으며 그 어떠한 grammar rule에 맞는 문장인지 판단해주고 아니면 에러를 출력, 맞으면 tree를 구성해 주도록 하였다.

* Precedence directives

각각의 token에 대해 우선순위를 정해주어야 conflict가 발생할 때 어떤 것을 우선적으로 수행해야하나 정할 수 있다. 그래서 cm.y 의 declaration 부분에서 token을 정의해 주어 우선순위를 정의해 주었다. 밑에 나올수록 우선순위가 높다는 뜻이므로 우선순위가 높은 것은 밑에서, 낮은 것은 위에서 정의해 주었다. 또 left assoc가 기본인데, right assoc를 하는 것은 없으므로 별다른 조치를 취하진 않았다.



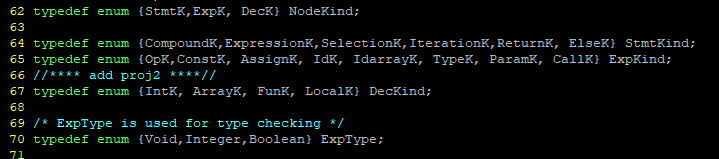
* New variable, function and type

cm.y에서는 savedName, LIneNo, Size, nowlevel이라는 변수를 새로 추가하였다. savedName과 lineNo는 저장하고 있는 동안 다른 이름과 lineno를 새로 저장해야 할 수 도 있으므로 배열로 넉넉하게 5000을 잡아 주었고, 현재 몇번째 index에 저장했나 확인하기 위해 now\_level을 선언, stack처럼 만들어 주었다.

savedSize에서는 변수, 그 중에서도 배열을 정의할 때 배열의 사이즈를 저장해 놓아야 하므로 그것을 위해 만들어 두었다.

global.h에서는 새로운 node type과, 각 node 의 kind를 알맞게 추가해 주었다. 먼저 선언부를 다루기 위하여 DecNode를 추가하였고, 이를 위해 util.c/h에 newDecNode 함수를 구현하였다. stmt, exp, dec kind에서는 각각에 맞는 걸 추가해 주었는데, 먼저 stmt kind에서는 교재에 나와 있는 6가지의 stmt에 대해 만들어 주었고, dec kind에서는 func인지 var인지, 만약 var이면 local인지 int인지 array인지에 대해 정의해 주었다. exp에서는 op인지 const인지 id인지 array id인지 type인지 parameter인지 call 인지 구분할 수 있게 정의해 주었다.

또 max child number을 5로 수정해 주었다.



scan.h에서는 beforeToken에 대해 정의해 두었는데, 이것은 token화 할 때 현재 토큰만 저장하지 않고 이전 토큰도 저장해 tree를 만들 때 각각의 lexeme이 무엇인지 확인하게끔 만들어 주었다. 그리고 알맞게 저장하기 위하여 cm.l의 getToken 부분에서 tokenString에 현재 토큰을 넣기 전에 tokenString이 가지고 있는 정보를 beforeToken에 넣어주었다.

* Print tree

각 translation rule 부분에서는 각각 rule에 왔을 때 어떻게 트리를 생성하거나, 붙여야 하는지 만들어 주었다.

책에 명시된 grammar rule의 순서대로 간략히 설명하겠다.

먼저 declaration part에서는 부모 노드가 fun dec인지 var dec인지 구분하고 그 node kind에 대해 정보를 담고 있다. 그 노드들은 자식들에게 부모의 type, 부모의 이름, 나머지 parameter나 array일시 배열 크기를 저장시켜 주었다.

parameter part에서는 , 만약 parameter가 없다면 자식을 가지지 않게 해 주고, print tree 부분에서 자식이 있는지 없는지 확인하여 있으면 출력, null이면 parameter가 없다고 출력해주게끔 하였다.

param list처럼 여러 개가 들어 올 수 있는 상황에선 sibling으로 연결해 주었다. param 부분에선 그 parameter의 타입과 id를 자식으로 가지게 해 주었다.

compound stmt에선 선언부와 statement list를 자식으로 가지게 해 주었다.

그 자식 부분인 local dec에선 여러 개의 변수를 가질 수 있으므로 local dec이 있다면 먼저 노드를 만들고, 여러개의 변수가 있다면 그 노드를 sibling으로 연결, 각각의 노드에서 그 변수의 타입과 id를 자식으로 가지게 해 주었다.

statement list역시 여러 개가 올 수 있으므로 sibling 처리를 해 주었다.

statement 부분에서는 새로운 노드의 구현 없이 바로 연결시켜 주었다.

expression stmt 부분에서도 expression이 있다면 그것을 바로 연결시켜 주고, 없다면 null로 정의해 주었다.

selection 부분에서는 node를 새로 만들고 조건부와 if일때의 statement, else가 있다면 그 때의 statement를 새로 만든 노드의 자식으로 넣어 주었다.

iteration 부분에서는 새로 노드를 만든 다음 조건부와 statement를 자식으로 넣어 주었다.

return 부분에서도 새로 노드를 만든 다음 expression 부분을 자식으로 넣어 주었다.

Assign 부분에서는 lvalue와 rvalue에 대해 각각 자식으로 넣어 주었고, simple expression이면 그냥 대입했다.

var 부분에서는 자신의 lexeme을 저장하고, 혹시 배열이라면 lexeme과 배열의 크기를 자식으로 넣어 주었다.

simple expression/ additive, mul 부분에서는 0, 1번째 자식에 비교할 것/ 계산할 것을 두고 op를 본인의 노드에 저장하였다. 즉 본인이 op를 가지고 있고, 그 op로 계산 당하는 것들은 0, 1번 자식에 들어가 있다.

relop, addop, mulop 부분에서는 위의 노드가 자신의 op가 무엇인지 알 수 있게끔 op를 저장해 주었다. 하지만 실제로 트리에는 연결하지 않고 부모 노드에서 자신의 op를 직접 가져가서 사용할 수 있게 했다.

factor 부분에서는 rule에 있는 실제 값 부분을 바로연결해 주었다.

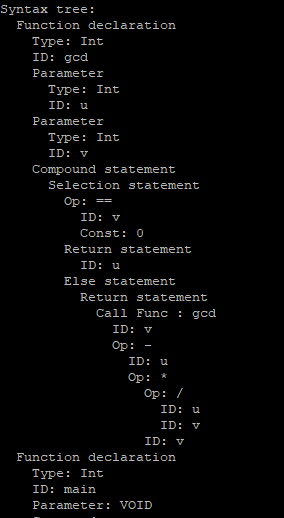
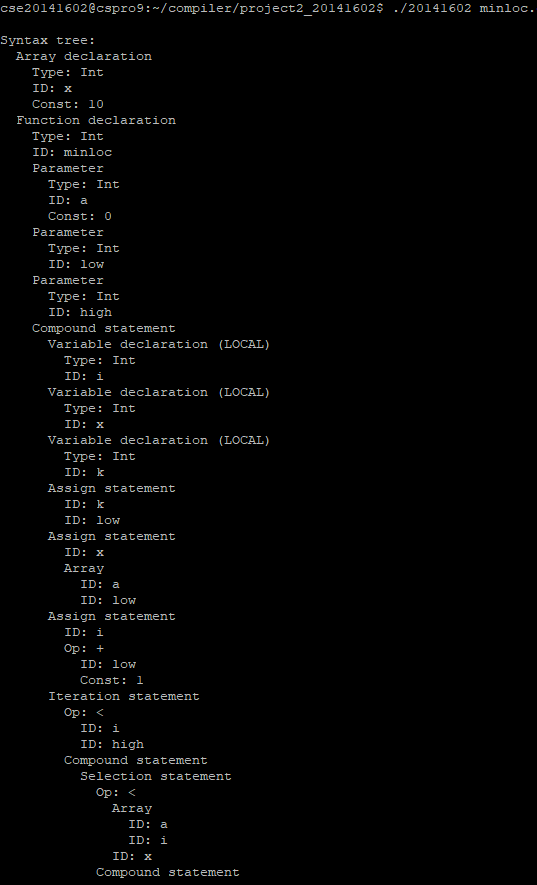
call 부분에서는 부르는 func/procedure의 id를 자식으로 가지게 해 두었다.

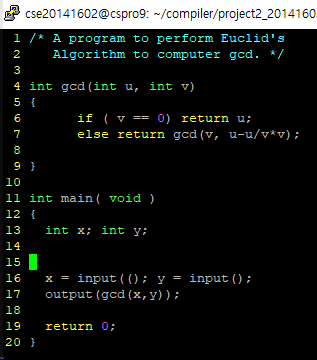
args 부분에서도 여러가지가 계속해서 나올 수 있는 arg list 부분에서는 그 부분끼리 sibling 처리를 해 주었다.

util.c의 print tree함수에서는 크게 총 세 가지의 경우로 나누어 처리해 주었다. 먼저 statement node일 경우에는 그 kind를 보고 어떤 statement인지 출력해 주었다. declaration node일 경우에도 비슷하게 어떤 것의 declaration인지를 출력해 주었다.

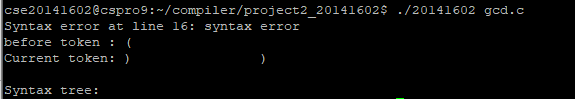
expression node에서는 각각의 kind를 보고 name이나 op, val을 출력해 주었다. type 부분에서는 node의 type 변수에 어떤 것이 저장되어있나 확인하고 int 혹은 void를 출력해 주었고, parameter 부분에서는 자식이 없다면, 즉 parameter가 없다면 null을 출력해 주고 자식이 있다면 자연스럽게 preorder처럼 순회하게끔 하고 자식까지 다 순회한 다음 sibling이 있으면 그것을 보게끔 만들어 주었다.

자식으로 들어가서 출력이 될 때는 indent가 한번씩 들어가고, 자식에서 나와 부모한테로 돌아갈때는 unindent 되며, sibling을 출력할 때엔 indent가 변화하지 않는다.

* 1. 시험
* ./20141602 gcd.c (일부)
* 
* ./20141602 minloc.c (일부)
* 
* ./20141602 gcd.c (syntax error를 넣었을 경우)



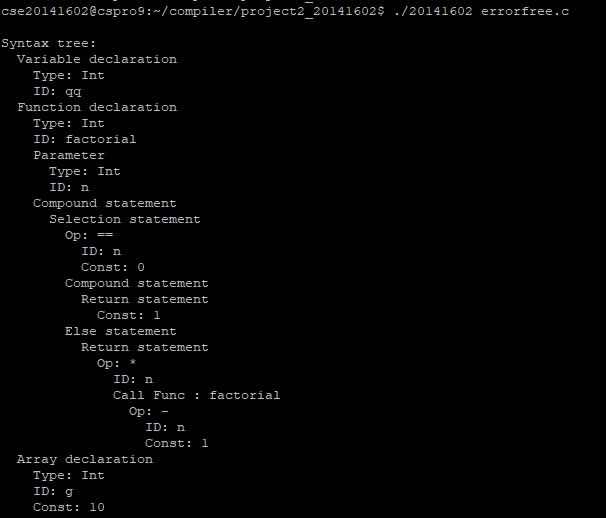
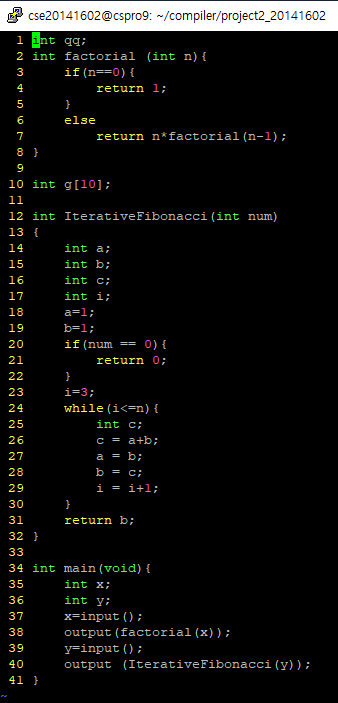
(16번째 줄 error)



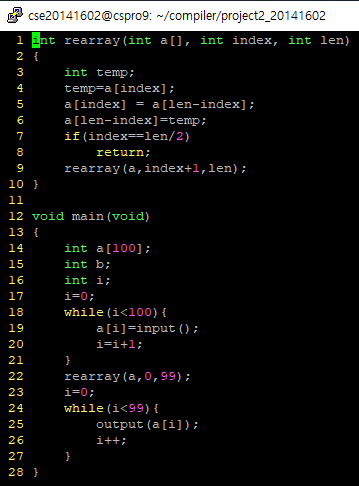
책에 나와있는 두 개의 예제에 대해서 시험을 해 보고, 또 직접 만든 코드에 대해 시험해 보았다. 예제 출력과 비교해 보며 child, sibling 등이 잘 들어간 것을 보았고 indent가 제대로 들어간 것을 보고 각각이 맞게 나뉘어진 것을 확인할 수 있었다.

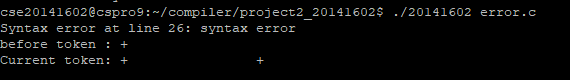
error case에서는 몇 번째 줄에서 syntax 에러가 났는지 확인할 수 있었고, 이전 토큰과 현재 토큰을 비교하며 어떤 부분에서 에러가 났는지 확인할 수 있었다.

* errorfree.c (일부)



* error.c (일부)





* 1. 평가

위 시험 내용에서 예시로 든 두 개의 소스코드 말고 다른 C- 소스 코드를 새롭게 작성해 넣어도 무리없이 원하는 결과를 출력하였다.

1. 기타
   1. 자체 평가

* translation rule을 만들 때 결과를 미리 생각하면서 rule을 만들었기 때문에 print tree할 때 별 어려움 없이 그 부분에 대해서만 출력을 해 주면 되었다. 그 부분에서 가독성을 높였다고 볼 수 있다.

* 1. 소감
* yacc이 어떻게 돌아가는지, 컴파일하면 이 많은 문법들의 syntax error를 어떻게 찾아내는지를 알 수 있는 프로젝트였다.