**컴파일러의 기초: Project 3 for Semantic Analysis**

**Due date : 11월 30일 금요일 23:59**

이번 프로젝트의 목적은 lex와 yacc utility를 이용하여 (C의 feature를 가감한; subc라 부른다)에 대한 semantic analysis를 구현하는 프로젝트이다.

프로젝트의 시작에 앞서 강의 교재(Ch 7. yacc, Ch 8. semantic analysis)와 주어진 subc문법에 대해서 철저하게 공부하는 것을 권장한다. 이번 프로젝트에서 구현한 사항들이 프로젝트4에서도 직접적으로 사용되기 때문에 전체적인 디자인이 매우 중요하다. 디자인에 대한 계획을 세웠다면 문서에 언급된 제한 사항을 만족하는 semantic analysis 를 구현하도록 한다. 구현 방법과 제한사항, 문법 등은 다음과 같다.

* **How to do this project3?**

Environment : 이전 환경과 동일, flex, bison 사용

Makefile : 이전에 주어진 형태에서 바꿔서 사용

**semantic check code**

subc.y 문법의 각 terminal과 nonterminal 사이의 적절한 위치에 action (C코드)를 삽입해서 아래 주어진 제한 사항들을 체크하고 만족하지 못 할 경우 화면으로 출력하도록 한다.

* **Restriction of this project**

이번 프로젝트는 다음과 같은 범위 정도로 체크하고, 또 구현하도록 한다.

각 제한사항에 대한 자세한 내용과 예시는 ppt를 참조하도록 한다.

Redeclaration of same variables at same scope

No Type Casting

Pointer Operation

Operation on Structures

Structure Declaration

Structure Pointer Declaration

Function

LValue Checking

* **Grammer**

프로젝트 2의 문법에서 anonymous structure declaration, binary의 \*, /, %연산 등이 제거 되고 pre-increment/decrement연산 등이 추가된 문법이다. 전체적으로는 이전보다 간략화 된 문법이다.

본인의 설계를 위해서 문법 구조를 변경해도 가능하나, 앞에서 언급된 제한 사항에 대해서는 모두 체크할 수 있도록 해야한다.

수정된 문법은 따로 grammar.txt파일로 제공한다.

TYPE : 'int', 'char'

VOID : 'void'

STRUCT : 'struct'

INTEGER\_CONST : integer

RETURN : 'return'

IF : 'if'

ELSE : 'else'

WHILE : 'while'

FOR : 'for'

BREAK : 'break'

CONTINUE : 'continue'

LOGICAL\_OR : '||'

LOGICAL\_AND : '&&'

RELOP : '<' '<=' '>=' '>' '>='

EQUOP : '==' '!='

CHAR\_CONST : character

STRING : string

INCOP : '++'

DECOP : '--'

STRUCTOP : '->'

'[' ']' '(' ')' '.' STRUCTOP : left

'!' INCOP DECOP : right

'\*' '/' '%' : left

'+' '-' : left

RELOP : left

EQUOP : left

'&' : left

LOGICAL\_AND : left

LOGICAL\_OR : left

'=' : right

',' : left

program

: ext\_def\_list

;

ext\_def\_list

: ext\_def\_list ext\_def

| /\* empty \*/

;

ext\_def

: type\_specifier pointers ID ';'

| type\_specifier pointers ID '[' const\_expr ']' ';'

| func\_decl ';'

| type\_specifier ';'

| func\_decl compound\_stmt

type\_specifier

: TYPE

| VOID

| struct\_specifier

struct\_specifier

: STRUCT ID '{' def\_list '}'

| STRUCT ID

func\_decl

: type\_specifier pointers ID '(' ')'

| type\_specifier pointers ID '(' VOID ')'

| type\_specifier pointers ID '(' param\_list ')'

pointers

: '\*'

| /\* empty \*/

param\_list /\* list of formal parameter declaration \*/

: param\_decl

| param\_list ',' param\_decl

param\_decl /\* formal parameter declaration \*/

: type\_specifier pointers ID

| type\_specifier pointers ID '[' const\_expr ']'

def\_list /\* list of definitions, definition can be type(struct), variable, function \*/

: def\_list def

| /\* empty \*/

def

: type\_specifier pointers ID ';'

| type\_specifier pointers ID '[' const\_expr ']' ';'

| type\_specifier ';'

| func\_decl ';'

compound\_stmt

: '{' local\_defs stmt\_list '}'

local\_defs /\* local definitions, of which scope is only inside of compound statement \*/

: def\_list

stmt\_list

: stmt\_list stmt

| /\* empty \*/

stmt

: expr ';'

| compound\_stmt

| RETURN ';'

| RETURN expr ';'

| ';'

| IF '(' expr ')' stmt

| IF '(' expr ')' stmt ELSE stmt

| WHILE '(' expr ')' stmt

| FOR '(' expr\_e ';' expr\_e ';' expr\_e ')' stmt

| BREAK ';'

| CONTINUE ';'

expr\_e

: expr

| /\* empty \*/

const\_expr

: expr

expr

: unary '=' expr

| or\_expr

or\_expr

: or\_list

or\_list

: or\_list LOGICAL\_OR and\_expr

| and\_expr

and\_expr

: and\_list

and\_list

: and\_list LOGICAL\_AND binary

| binary

binary

: binary RELOP binary

| binary EQUOP binary

| binary '+' binary

| binary '-' binary

| unary %prec ‘=’

unary

: '(' expr ')'

| '(' unary ')'

| INTEGER\_CONST

| CHAR\_CONST

| STRING

| ID

| '-' unary %prec '!'

| '!' unary

| unary INCOP

| unary DECOP

| INCOP unary

| DECOP unary

| '&' unary %prec '!'

| '\*' unary %prec '!'

| unary '[' expr ']'

| unary '.' ID

| unary STRUCTOP ID

| unary '(' args ')'

| unary '(' ')'

args /\* actual parameters(function arguments) transferred to function \*/

: expr

| args ',' expr

* **Examples**

|  |  |
| --- | --- |
| **>We check this file( compile failed)**  // test1.c file  int main() {  int a;  char a; // redeclaration error  int b;  int b; // redeclaration error  int c;  char d;  return 0;  }  **>Run**  ./subc test1.c  **>Results**  test1.c:3: error: declaration error  test1.c:5: error: declaration error | **>We check this file( compile succeeded)**  // test2.c file  int main() {  int a;  int b;  int c;  char d;  return 0;  }  **>Run**  ./subc test2.c  **>Results** |

* **제출 사항**

소스코드

subc.l, subc.y, hash.h, hash.c, makefile 등 작성한 소스코드

readme 파일

결과 보고서(4장 이내)

구현 방법, 구현 내용, 문제점 (각 구현 제한사항 별로 언급바람)에 대해서 작성한다.

project3\_학번.zip으로 압축하여 eTL을 통해 제출한다.

* **TA Contact**
  + 신창현 (301동 819호)
  + 전화 번호 : 02-880-1767
  + E-mail : schyun9212@altair.snu.ac.kr