21001715 컴파일러
Programming Assignment #3
Due: 2020년 5월 24일 23:59PM

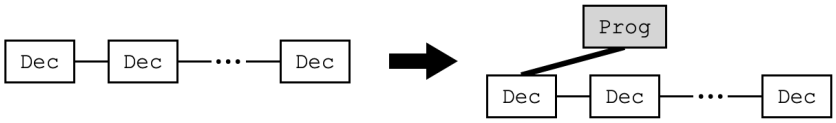
Abstract Syntax Tree 만들기

본 프로젝트에서는 PA2에서 만들었던 문법들을 바탕으로 AST를 만든다.

1. Your Mission

다음은 각 생성규칙마다 Subtree가 어떻게 만들어지는지를 보여준다. 이 때, 새로 생성되는 node는 회색 바탕으로, 새로 연결되는 것은 굵은 선으로 표시하였으며, 아무 action도 필요 없는 문법규칙은 밑줄로 표시하였다.

(1) `Program → DeclarationList`



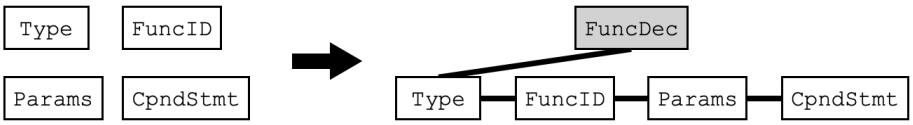
(2) `DeclarationList → DeclarationList Declaration`

(3) `DeclarationList → Declaration`

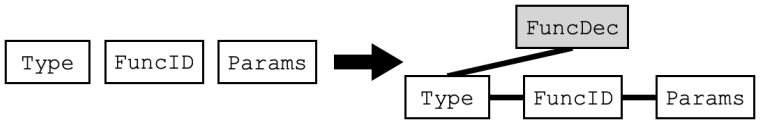
(4) `Declaration → VarDeclaration`

(5) `Declaration → FuncDeclaration`

(6) `FuncDeclaration → TypeSpecifier FuncID ( Params ) CompoundStmt`

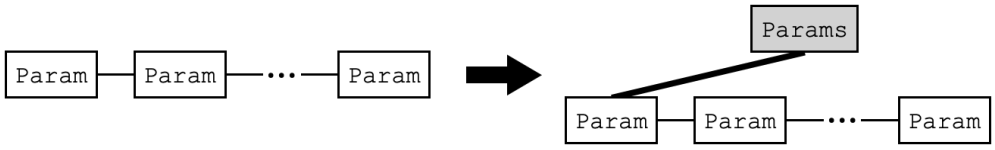


(7) `FuncDeclaration → TypeSpecifier FuncID ( Params ) ;`

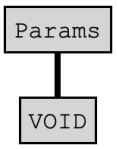


(8) `FuncID → ID` `ID` node 생성

(9) Params → ParamList



(10) Params → void

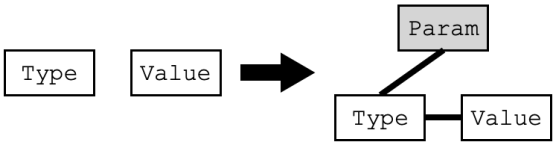


(11) Params → ε Params node 생성

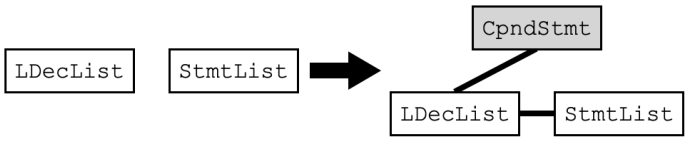
(12) ParamList → ParamList , Param

(13) ParamList → Param

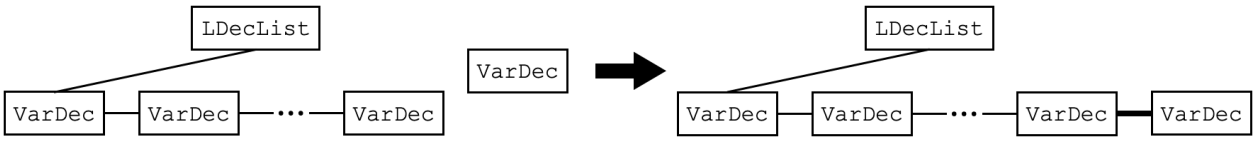
(14) Param → Type Value



(15) CompoundStmt → { LocalDeclarationList StmtList }

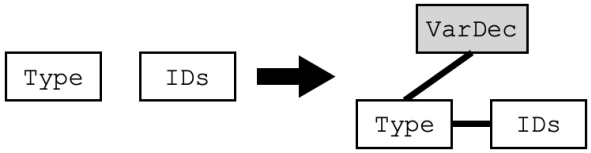


(16) LocalDeclarationList → LocalDeclarationList VarDeclaration



(17) LocalDeclarationList → ε LocalDeclarationList node 생성

(18) VarDeclaration → TypeSpecifier IDs ;

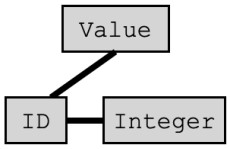


(19) TypeSpecifier → int | void | char | float 각각 type의 node가 생긴다.

(20) IDs: IDs , Value

(21) IDs: Value

(22) `Value → ID [ INTEGER ]`



(23) `Value → ID` `ID` node 생성

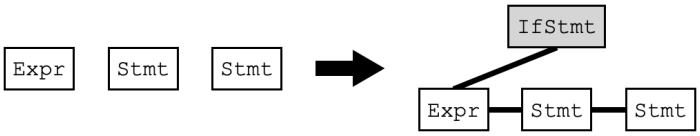
(24) `StmtList → StmtList Stmt` (16)과 비슷하게 하면 됨

(25) `StmtList → ε` (17)과 비슷하게 하면 됨

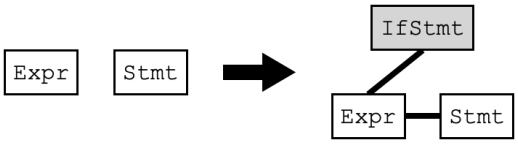
(26) `Stmt → MatchedStmt | OpenStmt`

(27)
```
MatchedStmt → ExprStmt
            | ForStmt
            | WhileStmt
            | ReturnStmt
            | CompoundStmt
            | BreakStmt
            | SwitchStmt
```

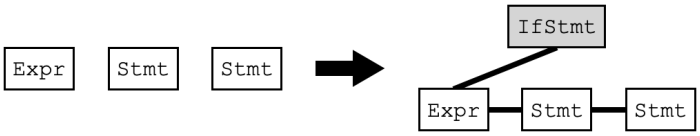
(28) `MatchedStmt → if ( Expr ) MatchedStmt else MatchedStmt`



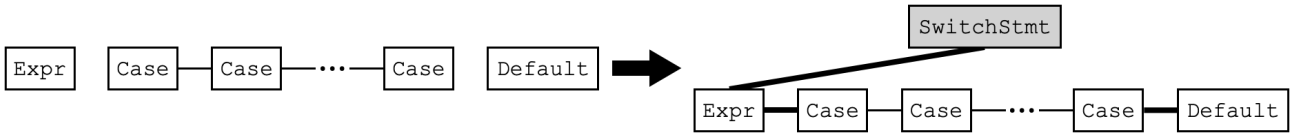
(29) `OpenStmt → if ( Expr ) Stmt`



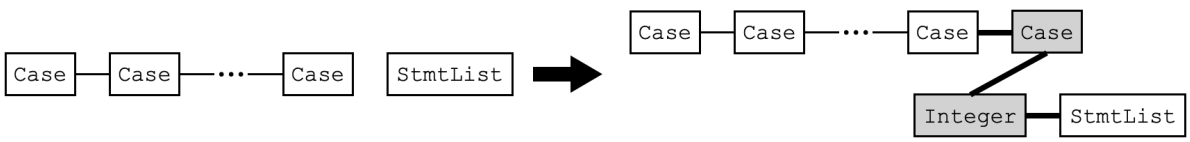
(30) `OpenStmt → if ( Expr ) MatchedStmt else OpenStmt`



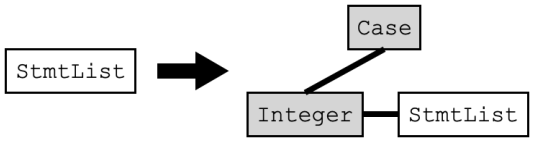
(31) `SwitchStmt → switch ( Expr ) { CaseList DefaultCase }`



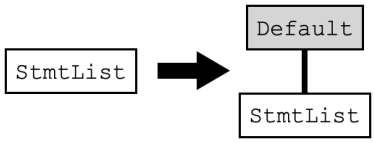
(32) `CaseList → CaseList case INTEGER : StmtList`



(33) `CaseList → case INTEGER : StmtList`

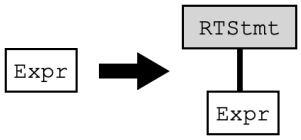


(34) `DefaultCase → default : StmtList`



(35) `DefaultCase → ε` `Default` node 생성

(36) `ReturnStmt → return Expr ;`



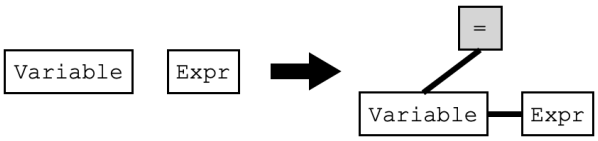
(37) `ReturnStmt → return ;` `ReturnStmt` node 생성

(38) `BreakStmt → break ;` `BreakStmt` node 생성. 이 때, `break` 문은 반복문이나 `switch` 문 안에서만 쓰여야 한다. 그 외에 쓰였을 때는 문법 error를 발생시켜야 하는데, 이를 감지하는 기능을 추가한다.

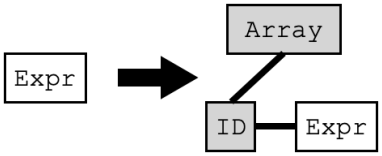
(39) `ExprStmt → Expr ; | ;`

(40) `Expr → AssignExpr | SimpleExpr`

(41) `AssignExpr → Variable = Expr` (+=, -=, *=, /=, %= 마찬가지로 한다.)

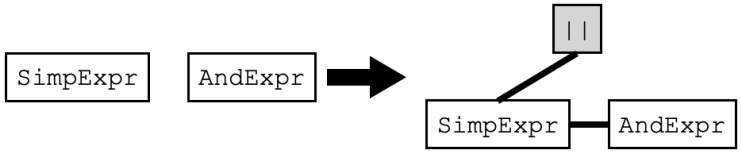


(42) `Variable → ID [ Expr ]`



(43) `Variable → ID` `ID` node 생성

(44) `SimpleExpr → SimpleExpr || AndExpr`



(45) `SimpleExpr → AndExpr`

(46) `AndExpr → AndExpr && RelExpr` (44)와 비슷하게 함

(47) `AndExpr → RelExpr`

(48) `RelExpr → RelExpr < AddExpr` (44)와 비슷하게 함, <=, >, >=, ==, != 모두 마찬가지로 함.

(49) RelExpr → AddExpr

(50) AddExpr → AddExpr + Term　　(44)와 비슷하게 함, - 도 마찬가지로 함.

(51) AddExpr → Term

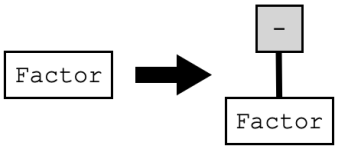
(52) Term → Term * Factor　　(44)와 비슷하게 함, /, %도 마찬가지로 함.

(53) Term → Factor

(54) Factor → (Expr)

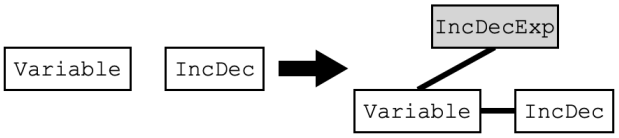
(55) Factor → FunctionCall

(56) Factor → - Factor

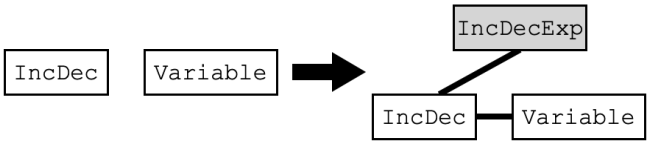


(57) Factor → Variable

(58) Factor → Variable IncDec



(59) Factor → IncDec Variable



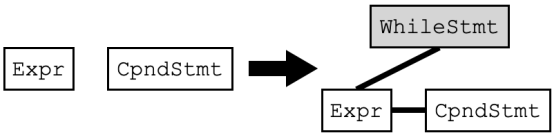
(60) Factor → NumberLiteral

(61) NumberLiteral → INTEGER　　INTEGER node 생성

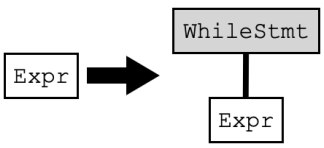
(62) NumberLiteral → REAL　　REAL node 생성

(63) IncDec → ++ | --　　각각 ++, -- node 생성

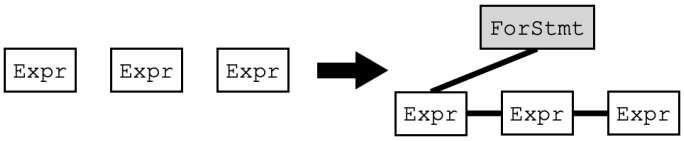
(64) WhileStmt → while (Expr) CompoundStmt



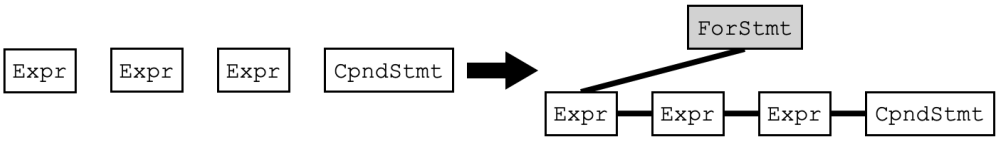
(65) WhileStmt → while (Expr) ;



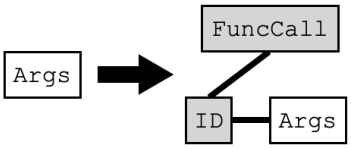
(66) ForStmt → for (Expr ; Expr ; Expr) ;



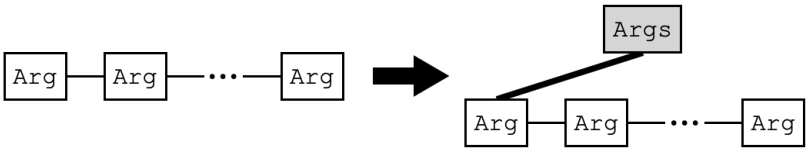
(67) `ForStmt → for ( Expr ; Expr ; Expr ) CompoundStmt`



(68) `FunctionCall → ID ( Arguments )`



(69) `Arguments → ArgumentList`



(70) `Arguments → ε` `Args` node 생성

(71) `ArgumentList → ArgumentList , Expr`

(72) `ArgumentList → Expr`

2. 작성 방법

(0) PA2 파일 복사.
여러분들의 계정에 PA3 디렉토리와 Makefile, test.c, ast.h, util.a 파일이 있을 것이다. 여기에 PA2에서 만들었던 foo.l 와 bar.y 파일을 복사해온다. 현재 위치가 PA3 디렉토리일 때, 복사 방법은 다음과 같다.

```
$ cp ../PA2/foo.l .
$ cp ../PA2/bar.y .
```

(1) foo.l 수정
- 만일 수정 사항이 있을 경우, 수정 해도 무방함.

(2) bar.y 수정
- 맨 위에 #include "ast.h" 를 추가한다.
- 바로 밑에 STACK *stack; 을 추가하여, 전역변수 stack을 선언한다.
- main함수에 내용을 다음과 같이 수정한다:

```
extern FILE *yyin;
stack = initStack();
yyin = fopen(argv[1], "r");
yyparse();
fclose(yyin);
printAST(pop(stack));
return 0;
```

3. ast.h 설명

AST 구성을 위한 node 및 함수와 node들을 관리할 수 있는 stack을 관리하는 함수를 선언해 놓은 헤더파일로, 각각의 설명은 다음과 같다.

- ASTNode: AST의 node를 위한 구조체 타입.
- STACK: AST의 node를 관리할 수 있는 stack을 위한 구조체 타입
- ASTNode* makeASTNode(TKNUM tknum, TYPE t) : AST의 node를 만들고, 그 node의 주소를 반환하는 함수. 인자로 들어가는 TKNUM 타입의 값은 token number로 AST.h에서 typedef enum {…} TKNUM으로 정의된 원소들 중 하나를 넣으면 되며, TYPE 타입의 값은 AST.h에서 typedef enum {…} TYPE으로 정의된 원소들이다. 즉, 각 node의 성질에 따라 맞는 값을 넣으면 됨.
  - PA3에서는 첫 번째 인자가 _TYPE인 경우를 제외하고는 두 번째 인자를 모두 NO_TYPE으로 해도 됨. TypeSpecifier인 경우에 _TYPE을 쓰는데, 해당 토큰이 int, float, void이면 각각 TYPE_INT, TYPE_FLOAT, TYPE_VOID를 쓰면 됨.
- ASTNode* makeASTNodeID(char *id, TYPE t) : makeASTNode 와 마찬가지로 node를 만들고 그 node의 주소를 반환하는 함수이나, id를 나타내는 토큰을 위한 node에 쓰인다. 인자로는 id의 이름이 문자열로 들어간다.
- ASTNode* makeASTNodeOP(char *op, TYPE t) : makeASTNode 와 마찬가지로 node를 만들고 그 node의 주소를 반환하는 함수이나, 연산자를 나타내는 토큰을 위한 node에 쓰인다. 인자로는 연산자의 이름이 문자열로 들어간다.
- ASTNode* makeASTNodeINT(int n) : makeASTNode 와 마찬가지로 node를 만들고 그 node의 주소를 반환하는 함수이나, 정수 상수를 나타내는 토큰을 위한 node에 쓰인다. 인자로는 해당 정수가 들어간다.
- ASTNode* makeASTNodeREAL(float r) : makeASTNode 와 마찬가지로 node를 만들고 그 node의 주소를 반환하는 함수이나, 정수 상수를 나타내는 토큰을 위한 node에 쓰인다. 인자로는 해당 정수가 들어간다.
- ASTNode* getSibling(ASTNode* n) : node n의 sibling node를 반환하는 함수. 만일 sibling이 없으면 0 (NULL) 을 반환.
- ASTNode* getChild(ASTNode* n) : node n의 첫 번째 child node를 반환하는 함수. 만일 child가 없으면 0 (NULL) 을 반환

- `ASTNode* setSibling(ASTNode* l, ASTNode* r)` : node `r`을 node `l`의 sibling으로 연결해주고, `l`의 주솟값을 반환해주는 함수.
- `ASTNode* setLastSibling(ASTNode* l, ASTNode* r)` : node `r`을 node `l`의 마지막 sibling으로 연결해주고, `l`의 주솟값을 반환해주는 함수.
- `ASTNode* setChild(ASTNode* p, ASTNode* c)` : node `c`를 node `p`의 child로 연결해주고, `p`의 주솟값을 반환해주는 함수.
- `TKNUM getTkNum(ASTNode *n)` : node `n`의 token number를 반환하는 함수.
- `TYPE getType(ASTNode *n)` : node `n`의 type을 반환하는 함수.
- `void printAST(ASTNode* head)` : node `head`를 root로 하는 subtree를 화면에 출력하는 함수.

다음은 stack 관련한 함수들이다.

- `STACK* initStack(void)` : stack을 초기화해주는 함수. stack을 사용하기 위해서는 보통 다음과 같이 선언과 함께 call하면 된다:
  `STACK *stack = initStack();`
- `void delStack(STACK* s)` : stack을 메모리에서 해제하는 함수. stack을 다 쓰고, 프로그램 끝내기 직전에 call 하면 된다
- `void push(STACK* s, ASTNode* n)` : stack `s`에 node `n`을 push하는 함수
- `ASTNode* pop(STACK *s)` : stack `s`에서 node를 pop하고, 그 pop한 node의 주소를 반환하는 함수.
- `void printStack(STACK *s)` : stack `s`의 내용을 화면에 출력하는 함수

4. 실행 예

test.c:

```
int prime(int p);
int main(void){
        int n;
        prime(n);
        return 0;
}
int prime(int p){
        int i;
        if(p <= 1)      return 0;
        for(i=2;i<p;i++){
                int r;
                r = p % i;
                if(r == 0)      return 0;
        }
        return 1;
}
```

위와 같은 소스코드가 있을 때, `./parser test.c` 를 실행하면 다음과 같은 결과가 나오게 하면 된다.

```
Program
    FunctionDec
        Type: int
        ID: prime
        Parameters
            Parameter
                Type: int
                ID: p
    FunctionDec
        Type: int
        ID: main
        Parameters
            Type: void
        CompoundStmt
```

```
        LocalDecList
            VariableDec
                Type: int
                ID: n
        StmtList
            FunctionCall
                ID: prime
                Arguments
                    ID: n
            ReturnStmt
                0
FunctionDec
    Type: int
    ID: prime
    Parameters
        Parameter
            Type: int
            ID: p
    CompoundStmt
        LocalDecList
            VariableDec
                Type: int
                ID: i
        StmtList
            IfStmt
                <=
                    ID: p
                    1
                ReturnStmt
                    0
            ForStmt
                =
                    ID: i
                    2
                <
                    ID: i
                    ID: p
                IncDecExp
                    ID: i
                    ++
                CompoundStmt
                    LocalDecList
                        VariableDec
                            Type: int
                            ID: r
                    StmtList
                        =
                            ID: r
                            %
                                ID: p
                                ID: i
                        IfStmt
                            ==
                                ID: r
                                0
                            ReturnStmt
                                0
            ReturnStmt
                1
```

5. 제출

- 5월 24일 일요일 23:59까지. 시계는 제출용 서버의 시계를 따른다.

- 여러분의 계정에 `PA3` 디렉토리가 있고, 이 안에 `Makefile`, `test.c`, `ast.h`, `util.a` 파일들이 있을 것이다. `bar.y` 파일의 내용을 바꾸면 된다.

- 만일 위의 파일들 중 어느 하나를 지웠으면 담당 교수에게 문의할 것.

- 하루씩 Delay 될 때마다 점수의 20%를 감점한다. (예를 들어, 이틀 Delay하고 8점을 받았으면 4.8점 획득)

- Copy는 해당 PA 0점. 본인 소스코드는 반드시 본인이 모두 타이핑 하여 만들어야 함. 오해를 방지하기 위하여 둘 이상이 같이 의논하여 짰더라도 최대한 달라보이게 짤 것! (변수 바꾸기, 주석 더 넣기, 들여쓰기 바꾸기 등 만으로는 안 됨)