C- 언어 파서 구현 보고서

github.com/squareyun

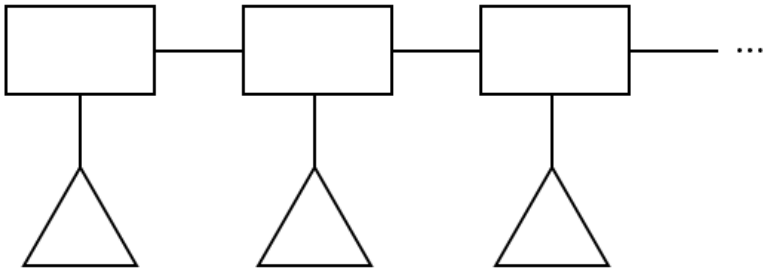
1. EBNF 변경 내용

recursive descent parser의 방법을 이용하여 파서를 구현하기 위해서는 EBNF로 변경하는 작업이 필요합니다. 교재에 나와있는 Grammer를 EBNF로 변경한 내용을 작성하였습니다.

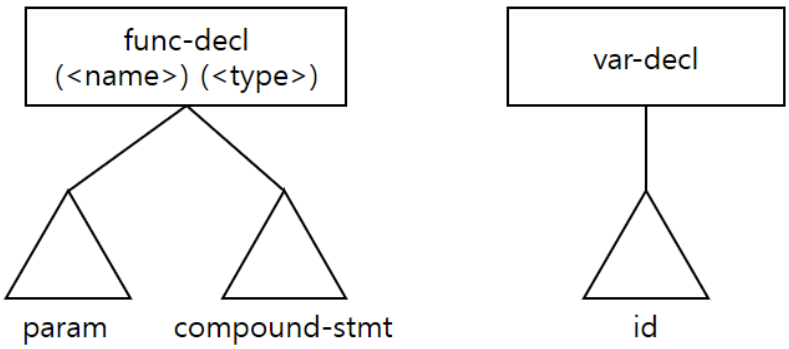
1. 변경 전후 동일
2. *declaration-list* → *declaration* { *declaration* }
3. 변경 전후 동일
4. 변경 전후 동일
5. 변경 전후 동일
6. 변경 전후 동일
7. 변경 전후 동일
8. *param-list* → *param* { **,** *param* }
9. *param* → *type-specifier* ***ID*** [**[ ]**]
10. 변경 전후 동일
11. *local-declarations* → *empty* { *var-declaraion* }
12. *statement-list* → *empty* { *statement* }
13. 변경 전후 동일
14. 변경 전후 동일
15. *selection-stmt* → ***if*** **(** *expression* **)** *statement* [***else*** *statement*]
16. 변경 전후 동일
17. 변경 전후 동일
18. 변경 전후 동일
19. *var* → *ID* [**[** *expression* **]**]
20. *simple-expression* → *additive-expression* [*relop additive-expression*]
21. 변경 전후 동일
22. *additive-expression* → *term* { *addop term* }
23. 변경 전후 동일
24. *term* → *factor* { *mulop factor* }
25. 변경 전후 동일
26. 변경 전후 동일
27. 변경 전후 동일
28. 변경 전후 동일
29. *arg-list* → *expression* { **,** *expression* }

2. syntax tree structure for C-

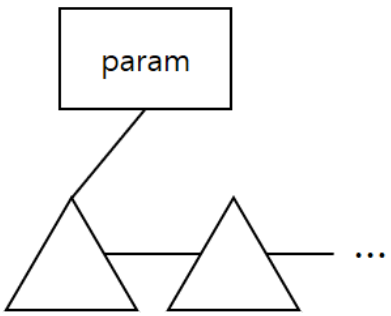
먼저, c- 언어의 각 문장들은 declaration의 연속으로 이루어집니다.



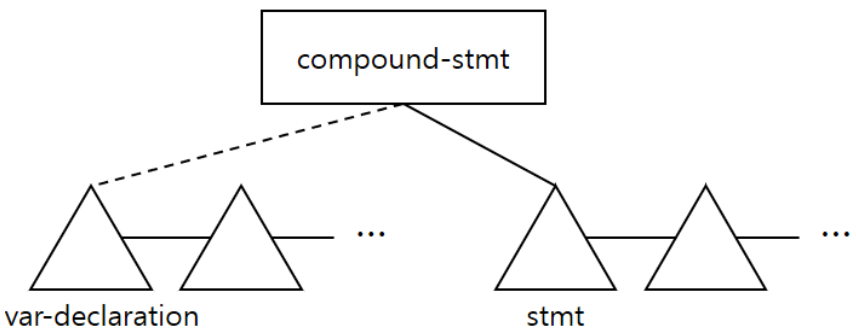
declaration은 func-declaration과 var-declaration으로 구성 되는데, fun-declaration은 두 개의 자식 노드를 가지며, var-declaration은 1개의 자식 노드를 가집니다. 각각 다음과 같은 구조를 가집니다.



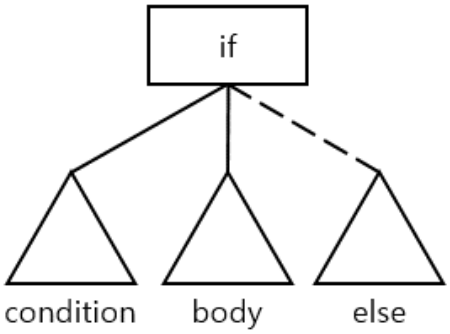
param은 한 개의 자식 노드로 구성되며, 그 자식 노드는 형제를 연속해서 가질 수 있습니다.



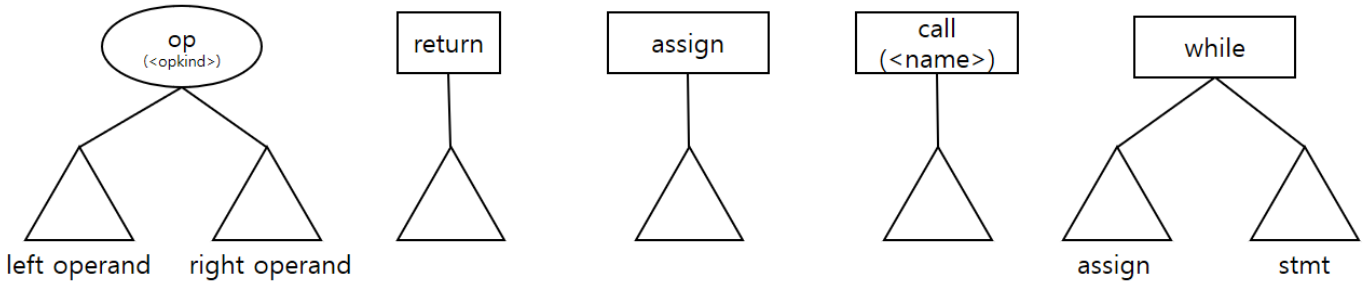
compound-stmt는 두 개의 자식 노드로 구성되나, 첫 번째 자식 노드는 없을 수도 있습니다. 첫 번째 자식 노드는 var-declaration으로 구성되며 이는 여러 개의 형제 노드를 가질 수 있습니다. 두 번째 자식 노드는 stmt로 구성되며 이 또한 여러 개의 형제 노드를 가질 수 있습니다.



selection-stmt는 condition, body, 그리고 else 부분으로 나뉩니다. else 부분은 없을 수도 있습니다.



op, return-stmt, assign, call, iteration-stmt 문장은 각각 다음과 같은 형태로 구성됩니다.



3. 프로그램 설명

3-1. function 설명

코드를 구성하는 함수 중 scanner에서 가져온 동일한 함수들의 설명은 생략하고, parser에 직접적인 관련이 있는 함수만 설명하고자 합니다. 입력, 출력에 대한 설명은 함수 정의로 대체하고 목적에서 자세히 설명 드립니다.

함수 정의	목적
TreeNode* newStmtNode(StmtKind kind);	Stmt 노드를 만드는 함수입니다. 입력으로 stmtkind라는 enum이 들어갑니다.
TreeNode* newExpNode(ExpKind kind);	Exp 노드를 만드는 함수입니다. 입력으로 expkind라는 enum이 들어갑니다. 노드의 type은 void로 초기화 되며, paramCheck라는 변수는 파라미터를 표현하는 노드인지 구별하기 위한 변수인데 FALSE로 초기화됩니다.
void match(TokenType expected);	match하고자하는 token이 입력으로 들어가고, 현재 읽어온 token과 동일하다면 getToken으로 다음 토큰을 읽습니다. 다르다면 에러를 출력합니다.
void syntaxError(char* message);	문장에 오류가 있을 때 오류 난 문장의 번호를 출력합니다.
ExpType type_checker(void);	Token의 type이 INT인지, VOID인지 확인하고 반환합니다. INT 또는 VOID라면 getToken으로 다음 토큰을 읽습니다. 따라서 이 함수를 호출하고 나면 다음 토큰을 다시 읽을 필요가 없습니다.

char* copyString(char* s);	문자열을 복사하는 함수입니다.
TreeNode* parse(void);	Parsing을 시작하는 함수입니다. declaration_list()를 호출함으로 parsing을 시작합니다.
TreeNode* declaration_list();	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. declaration을 호출하고, 추가적인 declaration이 존재할 시 형제 노드로 연결합니다.
TreeNode* declaration();	EBNF에 따라 구현하려면 토큰을 되돌리는 부분의 구현이 필요할 것입니다. 토큰을 되돌리지 않고, 이 함수에서 여러 가지 case를 나누어 var-declaration의 첫번째 경우, 두번째 경우, 그리고 fun-delaration의 경우 각각 구현하였습니다.
TreeNode* var_declaration(void);	위 declaration 함수에서 사용하는 것은 아니지만, 추후 EBNF를 위해 따로 구현을 하였습니다. (fun-declaration은 추후 필요하지 않아 구현하지 않았습니다.)
TreeNode* params(void);	type_checker을 확인한 후 void로 반환되는 경우 다음 토큰이 ')'인 경우 params의 선언이 끝나게 됩니다. 아니라면, 더 많은 파라미터가 존재한다는 의미이므로 param_list를 호출합니다.
TreeNode* param_list(ExpType type);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. param을 호출하고, 다음 토큰이 COMMA라면 계속하여 노드를 생성한 후 형제 노드로 이어 붙입니다.
TreeNode* param(ExpType type);	EBNF에 따라 Optional [_]을 적용하여 함수를 구성하였습니다. 파라미터라는 것을 인지하기 위해 노드의 paramCheck를 TRUE로 설정합니다.
TreeNode* compound_stmt(void);	첫 번째 자식 노드는 local_decl()를 호출하여 연결하고, 두번째 자식 노드는 stmt_list()를 호출하여 연결합니다.
TreeNode* local_decl(void);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. var_declaration()을 호출하고, 다음 토큰이 type으로 구성된다면 형제 노드로 계속 이어 붙입니다.
TreeNode* stmt_list(void);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. stmt()를 호출하고, 다음 토큰이 compound_stmt가 끝나는 즉, RCURY가 아닐 때까지 형제 노드로 계속 이어 붙입니다.
TreeNode* stmt(void);	stmt는 expression_stmt, selection_stmt, iteration_stmt, return_stmt, compound_stmt로 구성됩니다. 각 경우에 맞게 케이스를 나누어 해당 노드를 만드는 함수를 호출합니다.
TreeNode* expression_stmt(void);	compound_stmt가 끝나는 즉, 다음 토큰이 RCURY가 아니라면 expr()를 호출합니다.
TreeNode* selection_stmt(void);	if문을 구성하는 함수입니다. EBNF에 따라 Optional [_]을 적용하여 함수를 구성하였습니다. test 부분, then 부분, else 부분으로 나뉘는데 각 부분에 맞게 자식 노드를 차례대로 생성하여 연결합니다. test 부분은 expr()를 호출하며, then

	과 else 부분은 stmt()를 호출합니다.
TreeNode* iteration_stmt(void);	while문을 구성하는 함수입니다. 첫 번째 자식 노드는 expr()를 호출하여 연결하고, 두 번째 자식 노드는 stmt()를 호출하여 연결합니다.
TreeNode* return_stmt(void);	단순히 return;으로 끝나는 경우와 return expression;으로 끝나는 경우가 있습니다. 후자의 경우 expr()를 호출하여 첫 번째 자식 노드로 연결합니다.
TreeNode* expr(void);	expression을 이용하여 derivation 했을 때 토큰이 ID일 경우 var = expression 또는 call 함수를 호출할 때 입니다. 따라서 토큰이 ID일 경우 call()를 호출하여 노드를 생성합니다. 그 다음 토큰이 ASSIGN일 경우 var = expression 형태의 문장이 만들어 져야합니다. 따라서 첫 번째 자식 노드에 생성했던 노드를 연결하고, 두 번째 자식 노드에 expr()를 호출하여 연결합니다. 위의 모든 경우에 해당하지 않는다면 simple_expr()를 호출하여 반환합니다.
TreeNode* simple_expr(TreeNode* f);	EBNF에 따라 Optional [_]을 적용하여 함수를 구성하였습니다. 상위 단계로부터 처리된 결과를 받아와서 노드를 생성해야 하므로 입력으로 expression 단계의 노드를 받아옵니다. 토큰이 LT, LE, GT, GE, EQ, NE일 경우 해당 op를 처리하고 첫 번째 자식 노드에 상위 단계에서 받아온 노드를 연결하고 두 번째 자식 노드에 add_expr()를 호출하여 연결합니다.
TreeNode* add_expr(TreeNode* f);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. 토큰이 PLUS 또는 MINUS라면 term() 함수를 지속적으로 호출합니다.
TreeNode* term(TreeNode* f);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. 토큰이 MUL 또는 DIV라면 factor() 함수를 지속적으로 호출합니다.
TreeNode* factor(TreeNode* f);	문법에 여러가지 케이스가 존재합니다. 각 케이스에 맞게 함수를 호출하여 연결합니다.
TreeNode* call(void);	call 문법은 ID (args)입니다. ID를 소비한 후 단순히 args를 호출하는 것이 아닌, 문법에 따라 derivation 했을 때 만나는 각 경우를 모두 구현하였습니다.
TreeNode* args(void);	args가 empty가 될 수 있습니다. 토큰이 RPAREN 즉, call이 끝나는 상황이라면 단순히 NULL을 반환하고 아니라면 args_list()를 호출합니다.
TreeNode* args_list(void);	EBNF에 따라 Repetition {_}을 적용하여 함수를 구성하였습니다. 토큰이 COMMA라면 expr() 함수를 지속적으로 호출하여 형제 노드로 연결합니다.
static void printSpaces(void);	출력을 위한 보조 함수입니다. 전역 변수인 indentno에 따라 공백(띄어쓰기)를 출력합니다.
char* typeName(ExpType type);	노드에 저장했던 토큰 type을 입력으로 받아 해당 문자열을 반환하는 함수입니다.

void printTree(TreeNode* tree);	syntax tree를 출력하는 함수입니다. 매크로 함수인 INDENT, UNINDENT와 출력 보조 함수 printSpaces를 이용해 트리 구조의 출력 결과물을 생성합니다. 제가 적용했던 syntax tree structure에 따라 출력 결과물이 나오도록 구성하였습니다.

3-2. sample 실행 결과

sample(1.c)를 실행한 결과물입니다. 2단으로 캡처하였습니다.

```
Syntax tree:
  Declare function: gcd, type: int
    params:
      Declare variable: u, type: int
      Declare variable: v, type: int
    Function Body:
      if:
        Condition:
          Op: ==
              Id: v
              Const: 0
        Body:
          return:
            Id: u
        Else body:
          return:
            Call: gcd
              Args:
                Id: v
                Op: -
                    Id: u
                    Op: *
                        Op: /
                            Id: u
                            Id: v
                        Id: v
```

```
Declare function: main, type: void
  params:
    Declare variable: (null), type: void
  Function Body:
    Declare variable: x, type: int
    Declare variable: y, type: int
    assign:
        Id: x
        Call: input
          Args: nothing
    assign:
        Id: y
        Call: input
          Args: nothing
    Call: output
      Args:
        Call: gcd
          Args:
            Id: x
            Id: y
```

sample(2.c)를 실행한 결과물입니다. 3단으로 캡처하였습니다.

```
Declare array: x[10], type: int
Declare function: minloc, type: int
  params:
    Declare array: a[], type: int
    Declare variable: low, type: int
    Declare variable: high, type: int
  Function Body:
    Declare variable: i, type: int
    Declare variable: x, type: int
    Declare variable: k, type: int
    assign:
      Id: k
      Id: low
    assign:
      Id: x
      Id: a
        Id: low
    assign:
      Id: i
      Op: +
        Id: low
        Const: 1
    while:
      Condition:
      Op: <
        Id: i
        Id: high
      Body:
      if:
        Condition:
        Op: <
          Id: a
            Id: i
          Id: x
        Body:
          assign:
            Id: x
            Id: a
              Id: i
          assign:
            Id: k
            Id: i
      assign:
        Id: i
        Op: +
          Id: i
          Const: 1
    return:
      Id: k
```

```
Declare function: sort, type: void
  params:
    Declare array: a[], type: int
    Declare variable: low, type: int
    Declare variable: high, type: int
  Function Body:
    Declare variable: i, type: int
    Declare variable: k, type: int
    assign:
      Id: i
      Id: low
    while:
      Condition:
      Op: <
        Id: i
        Op: -
          Id: high
          Const: 1
      Body:
      Declare variable: t, type: int
      assign:
        Id: k
        Call: minloc
          Args:
            Id: a
            Id: i
            Id: high
            Id: i
      assign:
        Id: t
        Id: a
          Id: k
      assign:
        Id: a
          Id: k
        Id: a
          Id: i
      assign:
        Id: a
          Id: i
        Id: t
      assign:
        Id: i
        Op: +
          Id: i
          Const: 1
```

```
Declare function: main, type: void
  params:
    Declare variable: (null), type: void
  Function Body:
    Declare variable: i, type: int
    assign:
      Id: i
      Const: 0
    while:
      Condition:
      Op: <
        Id: i
        Const: 10
      Body:
      assign:
        Id: x
          Id: i
        Call: input
          Args: nothing
      assign:
        Id: i
        Op: +
          Id: i
          Const: 1
    Call: sort
      Args:
      Id: x
      Const: 0
      Const: 10
    assign:
      Id: i
      Const: 0
    while:
      Condition:
      Op: <
        Id: i
        Const: 10
      Body:
      Call: output
        Args:
          Id: x
            Id: i
      assign:
        Id: i
        Op: +
          Id: i
          Const: 1
```

3-3. sample (1.c) 의 AST