```
%{
/**********PROLOGUE AREA**********/
//THIS AREA WILL BE COPIED TO y.tab.c CODE

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include "node.c"


//Entry point of parse tree
NODE *head;
//String buffer
char buf[100];

extern FILE *yyin;
extern char *yytext;
extern int yylineno;
char * filename;

int yylex();

void yyerror(const char *str)
{
	fprintf(stderr, "%s:%d: error: %s '%s' token \n", filename, yylineno,
str, yytext);
}

int yywrap() {
  return 1;
}
```

yywrap함수가 없어서 error가 나길래 추가했습니다.

```
NODE* AddNT(NODE* a, NODE* b, int isChild) {
	if(isChild) InsertChild(a, b);
	else InsertSibling(a, b);
	return b;
}

NODE* AddT(char* name, NODE* a, char* Tval, int isChild) {
	NODE* b = MakeNode(strcat(strcat(strdup(name),": "),Tval));
	if(isChild) InsertChild(a, b);
	else InsertSibling(a, b);
	return b;
}

%}
```

Non-terminal이면 지금노드와 다음노드를 합치는 AddNT

Terminal이면 다음노드를 MakeNode를 통해 메모리 할당하고 지금노드와 합치는 AddT

```
/**********GRAMMAR AREA**********/
%union {
	int number;
	char *string;
	NODE *node;
}

//Tokens & Nonterms
%token <string> DEFINE
%token <string> INT VOID
%token <string> IF FOR
%token <string> CONTINUE
%token <string> OP_ASSIGN OP_INC OP_ADD OP_MUL OP_LOGIC OP_REL
%token <string> ID
%token <string> NUM
%token <string> LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET COMMA SEMICOLON


%start c_code
%type <node> c_code code define_header func_def func_arg_dec body statement
assign_stmt continue_stmt decl_list decl_init al_expr rel_expr inc_expr
variable value type clause init_stmt test_expr update_stmt
```

Expr token은 사용되지 않아서 삭제했습니다.

```
%%
//CFG

//todo: build parse tree in actions
c_code:
	code
					{
$$ = MakeNode(strdup("c_code")); head = $$;
AddNT($$, $1, 1);
					}

	| c_code code
					{
$$ = MakeNode(strdup("c_code")); head = $$;
AddNT($$, $1, 1);
AddNT($1, $2, 0);
					}
	;
```

Bottom up parsing이므로 마지막에 접근한 c_code가 트리의 head입니다.

기본적으로 $$ = MakeNode.. 로 시작해서 노드를 만듭니다.

그리고 terminal인가 여부에 따라 AddNT, AddT함수를 이용해 노드를 합칩니다.

```
code:
```

```
	define_header		{
$$ = MakeNode(strdup("code"));
AddNT($$, $1, 1);
                                        }

	|func_def	{
$$ = MakeNode(strdup("code"));
AddNT($$, $1, 1);
                                        }
	;

define_header:
	DEFINE ID NUM
                                        {
$$ = MakeNode(strdup("define_header"));
NODE* tmp;
tmp = AddT("DEFINE", $$, $1, 1);
tmp = AddT("ID", tmp, $2, 0);
tmp = AddT("NUM", tmp, $3, 0);
                                        }
	;

func_def:
	type ID LPAREN func_arg_dec RPAREN LBRACE body RBRACE
                                        {
$$ = MakeNode(strdup("func_def"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("ID", tmp, $2, 0);
tmp = AddT("LPAREN", tmp, $3, 0);
tmp = AddNT(tmp, $4, 0);
tmp = AddT("RPAREN", tmp, $5, 0);
tmp = AddT("LBRACE", tmp, $6, 0);
tmp = AddNT(tmp, $7, 0);
tmp = AddT("RBRACE", tmp, $8, 0);
                                        }
	;

func_arg_dec:
	decl_list
                                        {
$$ = MakeNode(strdup("func_arg_dec"));
AddNT($$, $1, 1);
                                        }
	;

body:
	clause
                                        {
$$ = MakeNode(strdup("body"));
AddNT($$, $1, 1);
                                        }
	|statement
```

```
                                    {
$$ = MakeNode(strdup("body"));
AddNT($$, $1, 1);
                                    }
       |body body
                                    {
$$ = MakeNode(strdup("body"));
AddNT($$, $1, 1);
AddNT($1, $2, 0);
                                    }
       ;

clause:
    FOR LPAREN init_stmt test_expr SEMICOLON update_stmt RPAREN LBRACE body
RBRACE
                                    {
$$ = MakeNode(strdup("clause"));
NODE* tmp;
tmp = AddT("FOR", $$, $1, 1);
tmp = AddT("LPAREN", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
tmp = AddNT(tmp, $4, 0);
tmp = AddT("SEMICOLON", tmp, $5, 0);
tmp = AddNT(tmp, $6, 0);
tmp = AddT("RPAREN", tmp, $7, 0);
tmp = AddT("LBRACE", tmp, $8, 0);
tmp = AddNT(tmp, $9, 0);
tmp = AddT("RBRACE", tmp, $10, 0);
                                    }

    |IF LPAREN test_expr RPAREN LBRACE body RBRACE
                                    {
$$ = MakeNode(strdup("clause"));
NODE* tmp;
tmp = AddT("IF", $$, $1, 1);
tmp = AddT("LPAREN", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
tmp = AddT("RPAREN", tmp, $4, 0);
tmp = AddT("LBRACE", tmp, $5, 0);
tmp = AddNT(tmp, $6, 0);
tmp = AddT("RBRACE", tmp, $7, 0);
                                    }

    |IF LPAREN test_expr RPAREN statement
                                    {
$$ = MakeNode(strdup("clause"));
NODE* tmp;
tmp = AddT("if", $$, $1, 1);
```

output.txt의 169번째 줄이 IF가 아니라 if로 출력되어서 if로 바꿨습니다.

```
tmp = AddT("LPAREN", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
```

```
tmp = AddT("RPAREN", tmp, $4, 0);
tmp = AddNT(tmp, $5, 0);
                                    }
       ;

statement:
    assign_stmt SEMICOLON
                                    {
$$ = MakeNode(strdup("statement"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("SEMICOLON", tmp, $2, 0);
                                    }

    |continue_stmt SEMICOLON
                                    {
$$ = MakeNode(strdup("statement"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("SEMICOLON", tmp, $2, 0);
                                    }

       |decl_list SEMICOLON
                                    {
$$ = MakeNode(strdup("statement"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("SEMICOLON", tmp, $2, 0);
                                    }

       |error SEMICOLON
       ;

init_stmt:
       assign_stmt SEMICOLON

                                    {
$$ = MakeNode(strdup("init_stmt"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("SEMICOLON", tmp, $2, 0);
                                    }

       |decl_list SEMICOLON
                                    {
$$ = MakeNode(strdup("init_stmt"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("SEMICOLON", tmp, $2, 0);
                                    }
       ;

update_stmt:
```

```
       inc_expr
                                    {
$$ = MakeNode(strdup("update_stmt"));
AddNT($$, $1, 1);
                                    }
       |decl_list
                                    {
$$ = MakeNode(strdup("update_stmt"));
AddNT($$, $1, 1);
                                    }
       ;

assign_stmt:
   variable OP_ASSIGN al_expr
                                    {
$$ = MakeNode(strdup("assign_stmt"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("OP_ASSIGN", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                    }
   ;

continue_stmt:
   CONTINUE
                                    {
$$ = MakeNode(strdup("continue_stmt"));
AddT("CONTINUE", $$, $1, 1);
                                    }
   ;

test_expr:
       rel_expr
                                    {
$$ = MakeNode(strdup("test_expr"));
AddNT($$, $1, 1);
                                    }
       ;

decl_list:
       decl_init
                                    {
$$ = MakeNode(strdup("decl_list"));
AddNT($$, $1, 1);
                                    }

       |decl_list COMMA variable
                                    {
$$ = MakeNode(strdup("decl_list"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("COMMA", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
```

```
                                    }

        |decl_list COMMA decl_init
                                    {
$$ = MakeNode(strdup("decl_list"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("COMMA", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                    }
        ;

decl_init:
        type variable
                                    {
$$ = MakeNode(strdup("decl_init"));
AddNT($$, $1, 1);
AddNT($1, $2, 0);
                                    }
        ;

al_expr:
        NUM
                                    {
$$ = MakeNode(strdup("al_expr"));
AddT("NUM", $$, $1, 1);
                                    }

        | variable
                                    {
$$ = MakeNode(strdup("al_expr"));
AddNT($$, $1, 1);
                                    }

        | al_expr OP_ADD al_expr
                                    {
$$ = MakeNode(strdup("al_expr"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("OP_ADD", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                    }

        | al_expr OP_MUL al_expr
                                    {
$$ = MakeNode(strdup("al_expr"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("OP_MUL", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                    }
        ;
```

```
rel_expr:
	value
                                        {
$$ = MakeNode(strdup("rel_expr"));
AddNT($$, $1, 1);
                                        }

	| rel_expr OP_REL rel_expr
                                        {
$$ = MakeNode(strdup("rel_expr"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("OP_REL", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                        }

	| rel_expr OP_LOGIC rel_expr
                                        {
$$ = MakeNode(strdup("rel_expr"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("OP_LOGIC", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
                                        }
	;

inc_expr:
	variable OP_INC
                                        {
$$ = MakeNode(strdup("inc_expr"));
AddNT($$, $1, 1);
AddT("OP_INC", $1, $2, 0);
                                        }
	;

value:
	variable
                                        {
$$ = MakeNode(strdup("value"));
AddNT($$, $1, 1);
                                        }

	| NUM
                                        {
$$ = MakeNode(strdup("value"));
AddT("NUM", $$, $1, 1);
                                        }
	;

variable:
	ID
                                        {
$$ = MakeNode(strdup("variable"));
```

```
AddT("ID", $$, $1, 1);
                                    }

       |variable LBRACKET RBRACKET
                                    {
$$ = MakeNode(strdup("variable"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("LBRACKET", tmp, $2, 0);
tmp = AddT("RBRACKET", tmp, $3, 0);
                                    }

       |variable LBRACKET NUM RBRACKET
                                    {
$$ = MakeNode(strdup("variable"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("LBRACKET", tmp, $2, 0);
tmp = AddT("NUM", tmp, $3, 0);
tmp = AddT("RBRACKET", tmp, $4, 0);
                                    }

       |variable LBRACKET al_expr RBRACKET
                                    {
$$ = MakeNode(strdup("variable"));
NODE* tmp;
tmp = AddNT($$, $1, 1);
tmp = AddT("LBRACKET", tmp, $2, 0);
tmp = AddNT(tmp, $3, 0);
tmp = AddT("RBRACKET", tmp, $4, 0);
                                    }
       ;

type:
       VOID
                                    {
$$ = MakeNode(strdup("type"));
AddT("VOID", $$, $1, 1);
                                    }
       | INT
                                    {
$$ = MakeNode(strdup("type"));
AddT("INT", $$, $1, 1);
                                    }
       ;

%%
```

Bottom up parsing임을 이용해 Terminal부터 Node를 만들고 $$를 통해 부모로 전달하면 부모에서 만들어진 Node를 사용할 수 있습니다.

```
/**********EPILOGUE AREAR AREA**********/
//THIS AREA WILL BE COPIED TO y.tab.c CODE
```

```
int main(int argc, char **argv )
{
	++argv, --argc; /* skip over program name */
	if ( argc > 0 )
		yyin = fopen( argv[0], "r" );
	else
		yyin = stdin;
	filename = argv[0];

	yyparse();

	//todo: print tree
	WalkTree(head);

	return 0;
}
```

Head가 parse tree의 시작이므로 WalkTree(head)하면 됩니다.