Homework 6 (20 points)

Abstract Syntax Tree (AST)

1. Extend the Yacc-based LALR(1) parser with an attribute grammar for creating the Abstract Syntax Tree representation of the input program; **(15 points)**

2. Verify the correctness of the AST by traversing it in pre-order and generating a new source program semantically equivalent with the original source code. **(5 points)**

Use the following data types and structures to represent the AST. The mapping of grammar non-terminals to the AST data structure is described in the table below.

```
typedef enum { PROGRAM, ASSIGN, IF, WHILE, STATEMENT, CONST, VAR, TYPE,
     EXPR, INT_CONST, REAL_CONST, BOOL_CONST, STRING_CONST, IDENTIFIER, OP
     } node_type;


typedef enum { PLUS, MINUS, MUL, DIV, MOD, LT, LE, GT, GE, EQ, NE, AND, OR
     } operator;


typedef struct _node {
      node_type type;
      union {
            int iValue; /* integer, true, false, compOp, addOp, mulOp */
            float fValue;           /* number */
            char *identifier;       /* identifier */
            /* list of BNF right-hand side symbols of nonterminal type */
            struct _node *body;
      };
      struct _node *next ;          /* decl-list, stmt-list */
} node;
```

	type	***iValue***	***fValue***	***identifier***	***body***	***next***
program	*PROGRAM*				*1. decl-list* *2. statement*	
statement	*ASSIGN*				*1. IDENTIFIER* *2. index (optional)* *3. expr*	
statement	*IF*				*1. expr* *2. statement* *3. else-part (optional)*	
statement	*WHILE*				*1. expr* *2. statement*	
statement	*FOR*				*1. IDENTIFIER* *2. expr* *3. expr* *4. statement*	
statement	*READ* *WRITE*				*io-list*	
statement	*STATEMENT*				*stmt-list*	
decl-list	*declaration*					*decl-list*
declaration	*CONST*				*1. IDENTIFIER* *2. expr*	
declaration	*VAR*				*1. IDENTIFIER* *2. index (optional)* *3. expr*	
type	*TYPE*			***IDENTIFIER***		
index	*expr*				*expr*	
else-part	*statement*				*statement*	
io-list	*IDENTIFIER* *STRING_CONST*				*1. IDENTIFIER / STRING_CONST* *2. index (optional)*	*io-list*
stmt-list	*statement*					*stmt-list*
expr	*EXPR*				*1. expr* *2. compOp, addOp, mulOp (optional)* *3. expr (optional)*	
simpleExpr	*expr*				*expr*	
term	*expr*				*expr*	
factor	*INT_CONST*	***NUMBER***				
factor	*REAL_CONST*		***NUMBER***			
factor	*BOOL_CONST*	***TRUE, FALSE***				
factor	*IDENTIFIER*			***IDENTIFIER***		
factor	*expr*				*expr*	
compOp, addOp, mulOp	*OP*	*operator*				