Compiler Construction

IT794

Innovative Assignment

NIRMA UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED 'A' GRADE

by

Darshan Maradiya (17bce057)

Mayank Agal (17bce059)

Guided by

Prof. Monika Shah

Contents

Lucy : Compiler Definition Table

Source Language	*Lucy - Own Language (It's actually LuC that stands for Language that is User friendly and similar to C language. It will be more like C but will also have some user-friendly Syntax like Python have. Also "Lu C" sounds "Lucy" – The Early Human Species, we can relate it with C language here.)*	
Tokens	**Data Type (minimum 2)**	int, double, char, string, boolean, float
	Keywords	if, elif, else, for, while, int, double, char, string, boolean, continue, main
	Operators	Arithmetic, Relational, Logical, Bitwise, Assignment Swap Operator : <==> to swap the values of variables
	Constants	**Integer constants:** Any permutation of 0-9 numbers **String constants:** Any permutation of alphabets, numbers and special symbols inside double quotes **float constants:** Any permutation of 0-9 numbers and dot(' . ') as floating point, but precision after floating point up to 7 digits **double constants:** Any permutation of 0-9 numbers and dot(' . ') as floating point, but precision after floating point up to 15 digits **Boolean constants:** false, true **char constants:** Any one character from alphabets, number and special symbols (does not supports escape sequences)
	Control construct	if … elif … else
	Loop construct	for, while
	Comments	// /*….*/
	Special symbols	~ : , / ? > . < , }]) { [(_ - + = * & ^ ! % \ |

Operators

Precedence	Operator	Type	Associativity
17	()	Parenthesis	Left to Right
16	++	Unary post-increment	Right to Left
	--	Unary post-decrement	
15	++	Unary pre-increment	Right to Left
	--	Unary pre-decrement	
	+	Unary plus	
	-	Unary minus	
	!	Unary logical negation	
	~	Unary bitwise complement	
14	!	Unary factorial	Left to Right
13	**	Unary exponential	Right to Left
12	*	Multiplication	Left to Right
	/	Division	
	%	Modulus	
11	+	Addition	Left to Right
	-	Substraction	
10	<<	Bitwise left shift	Left to Right
	>>	Bitwise right shift	
9	<	Relational less than	Left to Right
	>	Relational greater than	
	<=	Relational less than or equal	
	>=	Relational greater than or equal	
8	==	Relational is equal to	Left to Right
	!=	Relational is not equal to	
7	&	Bitwise AND	Left to Right
6	^	Bitwise exclusive OR	Left to Right
5	|	Bitwise inclusive OR	Left to Right
4	&&	Logical AND	Left to Right
3	||	Logical OR	Left to Right
2	?:	Ternary conditional	Left to Right
1	=	Assignment	Right to Left
	+=	Addition assignment	
	-=	Substraction assignment	
	*=	Multiplication assignment	
	/=	Division assignment	
	%=	Modulus assignment	

Syntax Rules

1) The program has syntax like following:
```
<datatype> main {<statements>}
or
<datatype> main {
<statements>
}
```
2) A block can contain either statements or constructs.
3) Each statement is compulsorily ended with new line, that is we cannot have more than one statement in one line.
4) Each construct (if, for, while) has body compulsorily written inside curly braces.
5) Loops support break and continue statements.
6) In if construct, it can have one if block mandatorily, multiple elif blocks, one else block.
7) In for construct, the syntax is similar to the for construct in C, but only difference is in statement termination by new line inside the body. Following will give more clear idea(also in while construct).

   In C:
```
for(int i = 0; i<n; i++)
{
        cnt++;
}
for(int i=0; i<n; i++)
        cnt++;
for(int i=0; i<n; i++) {cnt++;}

for(int i=0; i<n; i++) cnt++;
```
   These four are valid.

   In Lucy:
```
for(int i = 0; i<n; i++)
{
        cnt++
}
```
   is valid.
```
for(int i=0; i<n; i++)
        cnt++
```
   is invalid.
```
for(int i=0; i<n; i++) {cnt++}
```
   is invalid as statement is not terminating by new line.
```
for(int i=0; i<n; i++) cnt++
```
   is invalid as it isn't inside curly braces though it is following the statement termination rule
```
for(int i=0; i<n; i++){cnt++
}
```

is valid.

```
for(int i=0; i<n; i++) {}
```

is valid.

8) for loop statements can have only one statement. i.e. for(i=0, j=0; i<n; i++, j++) is invalid.
9) Other Syntax rules are similar to C.
10) Escape Sequence Characters are not supported.
11) There is only one method exist and that is print(). It is taking only one argument and that is any valid expression (strings are also included). As escape sequences are not supported, \n is not supported in strings. Hence print() function itself makes a newline after printing the expression. print() function always need an argument, hence to print only the new line, proper syntax will be : print(“”) and not this : print().
12) Single line comments starts with double slash(//) and Multiline comments are written inside “/*” and ”*/”, similar to C.

Semantic Rules

1) Implicit type casting is taking place similar to the C and necessary type checking are also done which is described in following few points.
2) Factorial Operator works with int/long/char and doesn’t work with float/double/string.
3) Increment/Decrement operator works only with numeric type and char (since it can be represented in ASCII) but doesn’t work with string type.
4) We can change the sign of numeric type data but not of string type data.
5) Logical operators cannot work with string.
6) Bitwise operator works only with int, long and char (since it can be represented in ASCII) but doesn’t work with float/double/string type.
7) Comparisons between numeric type and numeric type is allowed. Comparisons between string type and string type is allowed. Comparisons between numeric type and string type is not allowed.
8) Except “+” operator, no other arithmetic operator works on string. (Note that implicitly casting of other data type to string is taking place if one of the operands are string type).
9) Modulo operator works only with int/long/char but not with float/double/string.
10) Numeric type can be converted to string but not vice versa.
11) break and continue statements should be inside loops only.
12) Scope resolution is handled. Hence no declaration is allowed in the same scope.
13) There is no scope of global variables.
14) Variables declared in first statement are having same scope as body block has. i.e. for(int i=0; i<n; i++){} in this loop, variable i cannot be declared again inside body.

Working process of compiler

1) There are three files basically:
    a. Lucy.l
    b. Lucy.y
    c. mylib.h
2) Lucy.l file is working as a lexical analyser. It is identifying the tokens from the program and giving it to the syntax analyser.
3) Lucy.y file is working as a syntax analyser. It contains the grammar for the language and validates the syntax of the entire program at same time while parsing. Syntax analyser stops working when it founds the first syntax error.
4) Lucy.y includes mylib.h file in its headers which is the heart of the program execution.
5) mylib.h file contains necessary data structures and functions to handle the further process described in following points.
6) If entire program is syntactically correct after parsing is done, the meaningful AST is built in the form of different connected nodes by node making functions written inside mylib.h
7) Then there are node executing functions which starts executing the tree nodes in appropriate flow starting from the root node of the tree.
8) While executing the AST, the semantic checking like scope resolution, type compatibility, implicit type conversions, only declared variables can be used, declared but uninitialized variables should be assigned to default values, valid break and continue statements etc are checked. It shows proper Error message with line no if found the first such error and then it stops executing further.
9) The variables are declared and stored in symbol table during the execution of the AST. After the end of the scope, the variables are discarded from the symbol table.

Limitations and future work

1) The compiler doesn’t show proper syntax error messages
2) The compiler have no support of global variables, switch-case construct, any library import, any data structure, any built-in or user defined function except print().
3) The compiler is not platform independent as this only part of front end and there is no code generation phase after the generation of AST. There are functions which execute the AST and hence the program runs.
4) Proper freeing of memory allocation is not done after its use.
5) Double datatype is buggy.
6) Distinguishing between int and long type is not there.

The heart of the compiler: mylib.h

Structures, Unions, Global Variables and functions that are used to build AST and evaluate the program in proper flow are written in mylib.h file in C language and is explained as following.

Structures and Unions

union	varvalue	
int	ival	*holds int value*
float	fval	*holds float value*
double	dval	*holds double value*
char	cval	*holds char value*
char*	strval	*pointer to the string*
char	bval	*holds boolean value (0 or 1) inside char type (1 byte)*

struct	variable	
char*	name	*name of the variable*
char*	type	*datatype of variable*
varvalue*	value	*pointer to the varvalue union that holds the value of the variable*
variable*	next	*pointer to the next declared variable in symbol table (actually a linked list)*

struct	valueNode	
char*	type	*datatype of the value*
varvalue*	value	*pointer to the varvalue union that holds the value*

struct	symbolNode	
char*	name	*undeclared variable name*
char*	type	*datatype of the value to be assigned*
varvalue*	value	*pointer to the varvalue union that holds the value to be assigned*
expressionNode*	expression_node	*pointer to the expressionNode structure that is to be evaluated to get the value to be assigned*

struct	listNode	
symbolNode*	symbol_node	*pointer to the symbolNode that holds the data about the variable to be declared*
listNode*	next	*pointer to the next listNode structure that is to be declared next with same datatype*

struct	declarationNode	
char*	type	*datatype in which all the variables in listNodes are to be declared*
listNode*	next	*linked list of listNode that holds all the variables to be declared along with their data*

struct	assignmentNode	
char*	operation	*assignment operation type (=, +=, -=, *=, ...)*
char*	name	*variable name present on the left side of the operator*
expressionNode*	expression_node	*pointer to the expressionNode structure present on the right side*

struct	ternaryExpression	
expressionNode*	expression_node_1	*pointer to the expressionNode structure present on the left of Question Mark(?)*
expressionNode*	expression_node_2	*pointer to the expressionNode structure present between Question Mark(?) and Colon(:)*
expressionNode*	expression_node_3	*pointer to the expressionNode structure present on the right of Colon(:)*

struct	binaryExpression	
char*	operation	*binary operation type (+, -, *, /, %, &, &&, ^, ...)*
expressionNode*	expression_node_1	*pointer to the expressionNode structure present on the left side of the operator*
expressionNode*	expression_node_2	*pointer to the expressionNode structure present on the right side of the operator*

struct	unaryExpression	
char*	operation	*unary operation type (++, --, !, -, ...)*
expressionNode*	expression_node_1	*pointer to the expressionNode structure present on either side of the operator*

union	expression	
valueNode*	value_node	*pointer to the valueNode structure that holds some type of value*
char*	variable	*variable name*
ternaryExpression*	ternary_expression	*pointer to the ternaryExpression structure*
binaryExpression*	binary_expression	*pointer to the binaryExpression structure*
unaryExpression*	unary_expression	*pointer to the unaryExpression structure*
assignmentNode*	assignment_node	*pointer to the assignmentNode structure whose variable's value is to be required eventually*

struct	expressionNode	
char*	expression_type	*type of the expression*
expression*	expression	*pointer to the expression union that holds the pointer to particular type of expression*
valueNode*	value_node	*pointer to the valueNode structure that will hold the evaluated value of this expressionNode eventually*

struct	swapOperationNode	
char*	variable_1	*name of the variable name present on the left side of swap operator (<==>)*
char*	variable_2	*name of the variable name present on then right side of swap operator (<==>)*

union	statement	
declarationNode*	declaration_node	*pointer to the declarationNode struture*
assignmentNode*	assignment_node	*pointer to the assignmentNode struture*
expressionNode*	expression_node	*pointer to the expressionNode struture*
swapOperationNode*	swap_operation_node	*pointer to the swapOperationNode struture*
int	break_node	*holds the line no where the break statement is present*
int	continue_node	*holds the line no where the continue statement is present*

struct	statementNode	
char*	statement_type	*type of the statement*
statement*	statement	*pointer to the statement union that holds particular type of statement*

struct	ifBlock	
expressionNode*	condition	*pointer to the expressionNode structure that represents the condtion for if block*
statementSetNode*	body	*pointer to the statementSetNode structure (actually a linked list of statementSetNode) represents body*

struct	elifBlock	
expressionNode*	condition	*pointer to the expressionNode structure that represents the condtion for elif block*
statementSetNode*	body	*pointer to the statementSetNode structure (actually a linked list of statementSetNode) represents body*
elifBlock*	next	*pointer to the elifBlock structure that is actually the next elif block followed*

struct	elseBlock	
statementSetNode*	body	*pointer to the statementSetNode structure (actually a linked list of statementSetNode) represents body*

struct	optionalBlocks	
elifBlock*	elif_block	*pointer to the elifBlock structure*
elseBlock*	else_block	*pointer to the elseBlock structure*

struct	ifConstruct	
ifBlock*	if_block	*pointer to the ifBlock structure*
optionalBlocks*	optional_blocks	*pointer to the optionalBlock structure that holds the pointers for elifBlocks and elseBlock*

struct	forConstruct	
statementNode*	first_statement	*pointer to the statementNode struture represents the first statement in for construct*
expressionNode*	middle_expression	*pointer to the expressionNode struture represents the middle condition in for construct*
statementNode*	last_statement	*pointer to the statementNode struture represents the last statement in for construct*
statementSetNode*	body	*pointer to the statementSetNode structure (actually a linked list of statementSetNode) represents body*

struct	whileConstruct	
expressionNode*	condition	*pointer to the expressionNode structure that represents the condtion for while construct*
statementSetNode*	body	*pointer to the statementSetNode structure (actually a linked list of statementSetNode) represents body*

union	construct	
ifConstruct*	if_construct	*pointer to the ifConstruct*
forConstruct*	for_construct	*pointer to the forConstruct*
whileConstruct*	while_construct	*pointer to the whileConstruct*

struct	constructNode	
char*	construct_type	*type of the construct*
construct*	construct	*pointer to the construct union that holds particular type of construct*

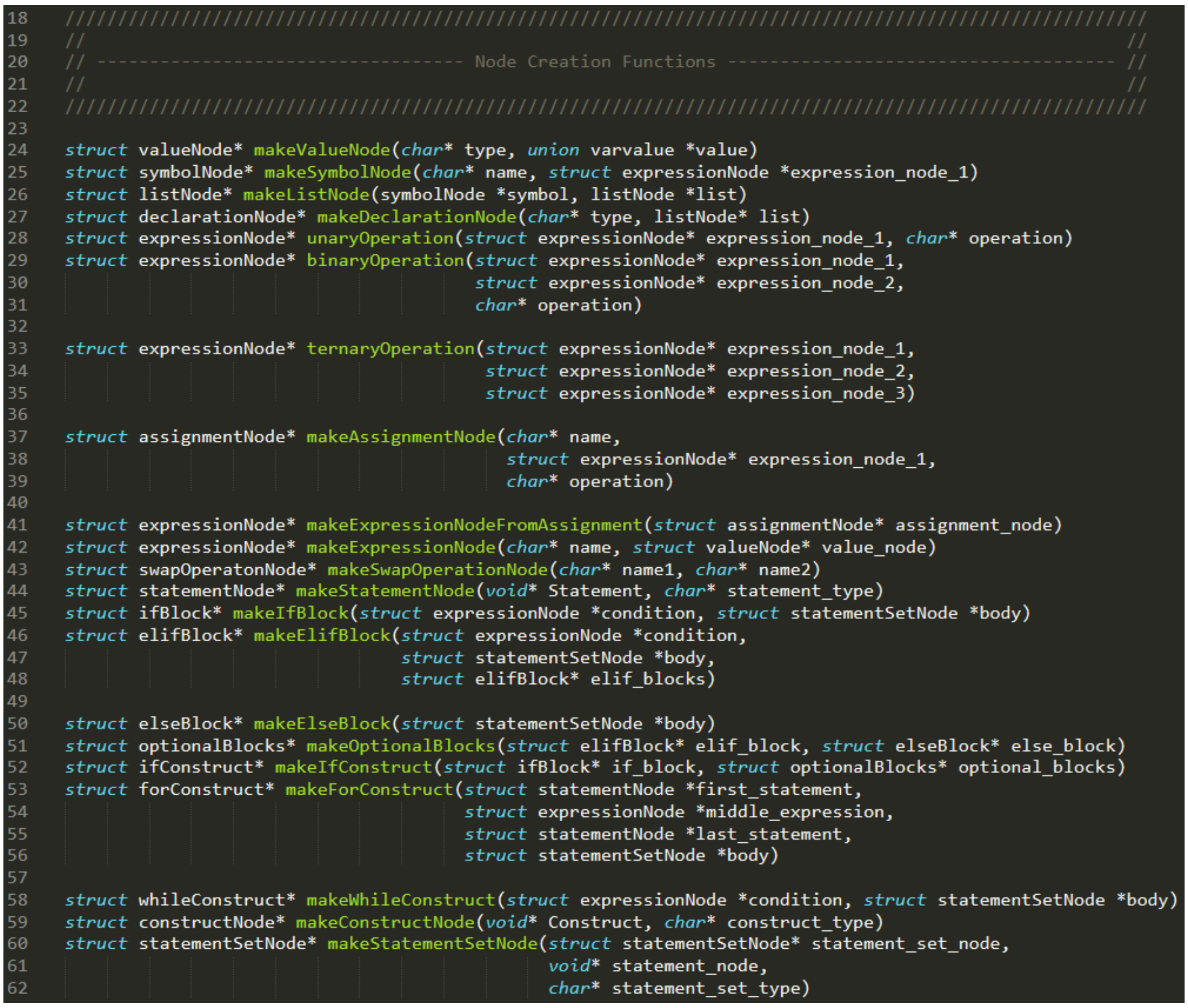
struct	statementSetNode	
char*	statement_set_type	*its is either "statementNode" or "constructNode", i.e. which pointer is to be followed from below two*
statementNode*	statement_node	*pointer to the statementNode structure*
constructNode*	construct_node	*pointer to the constructNode structure*
statementSetNode*	next	*pointer to the next statementSetNode that holds pointer to the next statement/construct followed*

Global variables

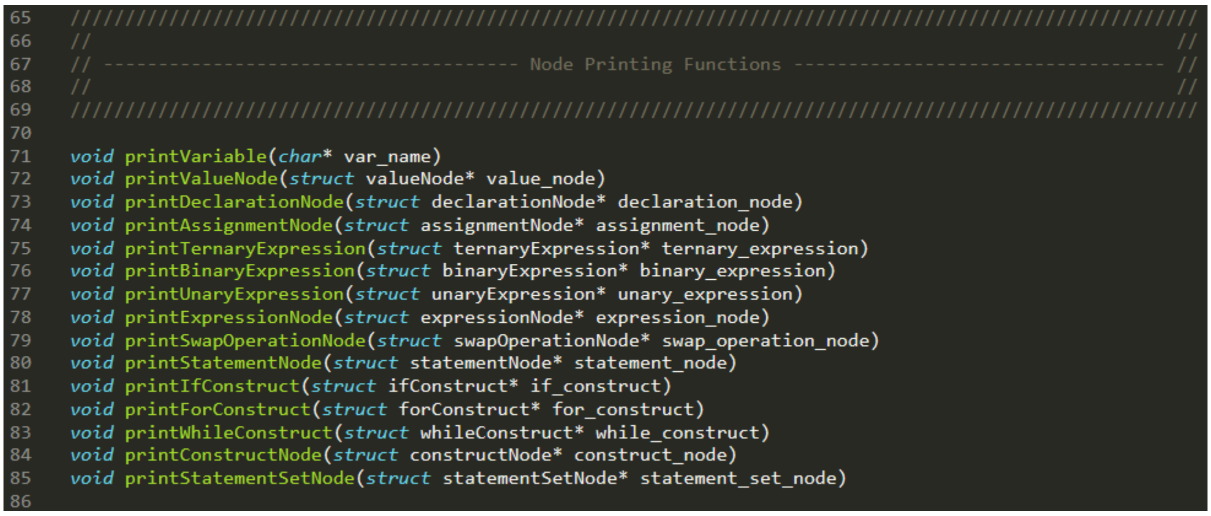
Type	Global Variable Name	Initial Value	Description
int	line	1	*to keep track of the current line no*
int	scope	0	*to keep track of the current scope value (higher in nested blocks)*
int	for_first_scope	-1	*scope of the first statement of for loop (-1 outside for loop)*
int	break_flag	0	*{0 : unset (invalid), 1 : do break, -1 : ready to break}*
int	continue_flag	0	*{0 : unset (invalid), 1 : do continue, -1 : ready to continue}*
variable*	start	NULL	*pointer to the head of the symbol table (linked list of variable structure)*

Functions

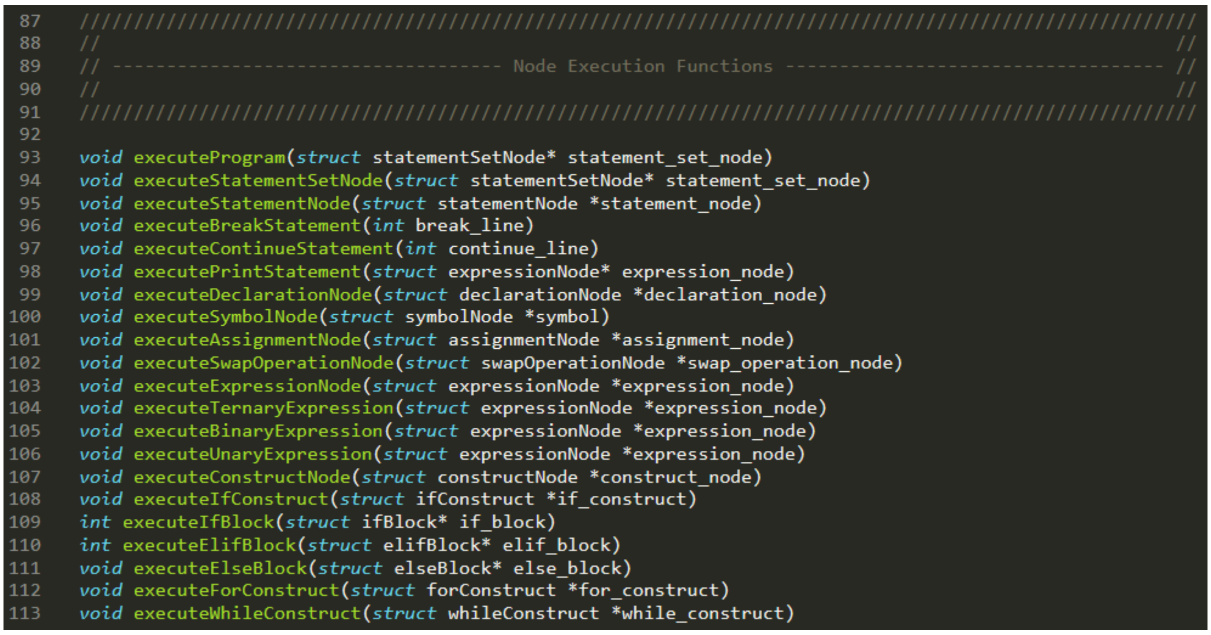
Following are the function names along with the arguments and its return type. The function names itself represents their purpose.



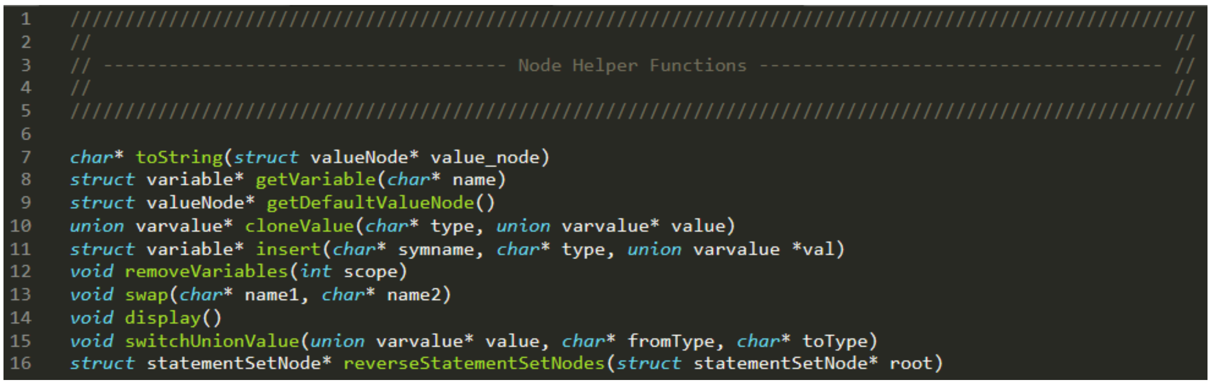
```
/////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                 //
// ---------------------------------- Node Creation Functions ------------------------------------ //
//                                                                                                 //
/////////////////////////////////////////////////////////////////////////////////////////////////////

struct valueNode* makeValueNode(char* type, union varvalue *value)
struct symbolNode* makeSymbolNode(char* name, struct expressionNode *expression_node_1)
struct listNode* makeListNode(symbolNode *symbol, listNode *list)
struct declarationNode* makeDeclarationNode(char* type, listNode* list)
struct expressionNode* unaryOperation(struct expressionNode* expression_node_1, char* operation)
struct expressionNode* binaryOperation(struct expressionNode* expression_node_1,
                                       struct expressionNode* expression_node_2,
                                       char* operation)

struct expressionNode* ternaryOperation(struct expressionNode* expression_node_1,
                                        struct expressionNode* expression_node_2,
                                        struct expressionNode* expression_node_3)

struct assignmentNode* makeAssignmentNode(char* name,
                                          struct expressionNode* expression_node_1,
                                          char* operation)

struct expressionNode* makeExpressionNodeFromAssignment(struct assignmentNode* assignment_node)
struct expressionNode* makeExpressionNode(char* name, struct valueNode* value_node)
struct swapOperatonNode* makeSwapOperationNode(char* name1, char* name2)
struct statementNode* makeStatementNode(void* Statement, char* statement_type)
struct ifBlock* makeIfBlock(struct expressionNode *condition, struct statementSetNode *body)
struct elifBlock* makeElifBlock(struct expressionNode *condition,
                                struct statementSetNode *body,
                                struct elifBlock* elif_blocks)

struct elseBlock* makeElseBlock(struct statementSetNode *body)
struct optionalBlocks* makeOptionalBlocks(struct elifBlock* elif_block, struct elseBlock* else_block)
struct ifConstruct* makeIfConstruct(struct ifBlock* if_block, struct optionalBlocks* optional_blocks)
struct forConstruct* makeForConstruct(struct statementNode *first_statement,
                                      struct expressionNode *middle_expression,
                                      struct statementNode *last_statement,
                                      struct statementSetNode *body)

struct whileConstruct* makeWhileConstruct(struct expressionNode *condition, struct statementSetNode *body)
struct constructNode* makeConstructNode(void* Construct, char* construct_type)
struct statementSetNode* makeStatementSetNode(struct statementSetNode* statement_set_node,
                                              void* statement_node,
                                              char* statement_set_type)
```



```
///////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// ------------------------------------- Node Printing Functions -------------------------------- //
//                                                                                               //
///////////////////////////////////////////////////////////////////////////////////////////////////

void printVariable(char* var_name)
void printValueNode(struct valueNode* value_node)
void printDeclarationNode(struct declarationNode* declaration_node)
void printAssignmentNode(struct assignmentNode* assignment_node)
void printTernaryExpression(struct ternaryExpression* ternary_expression)
void printBinaryExpression(struct binaryExpression* binary_expression)
void printUnaryExpression(struct unaryExpression* unary_expression)
void printExpressionNode(struct expressionNode* expression_node)
void printSwapOperationNode(struct swapOperationNode* swap_operation_node)
void printStatementNode(struct statementNode* statement_node)
void printIfConstruct(struct ifConstruct* if_construct)
void printForConstruct(struct forConstruct* for_construct)
void printWhileConstruct(struct whileConstruct* while_construct)
void printConstructNode(struct constructNode* construct_node)
void printStatementSetNode(struct statementSetNode* statement_set_node)
```



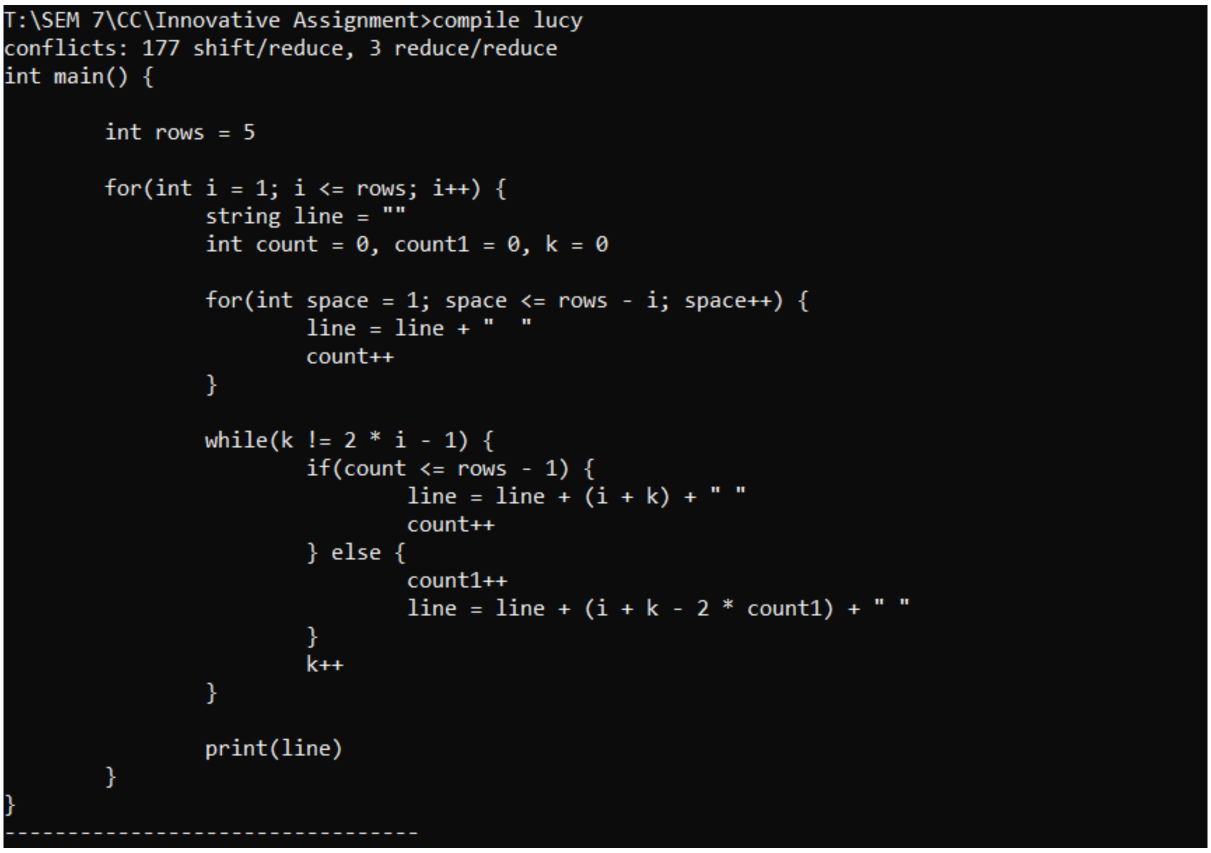
```
///////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// ----------------------------------- Node Execution Functions ---------------------------------- //
//                                                                                               //
///////////////////////////////////////////////////////////////////////////////////////////////////

void executeProgram(struct statementSetNode* statement_set_node)
void executeStatementSetNode(struct statementSetNode* statement_set_node)
void executeStatementNode(struct statementNode *statement_node)
void executeBreakStatement(int break_line)
void executeContinueStatement(int continue_line)
void executePrintStatement(struct expressionNode* expression_node)
void executeDeclarationNode(struct declarationNode *declaration_node)
void executeSymbolNode(struct symbolNode *symbol)
void executeAssignmentNode(struct assignmentNode *assignment_node)
void executeSwapOperationNode(struct swapOperationNode *swap_operation_node)
void executeExpressionNode(struct expressionNode *expression_node)
void executeTernaryExpression(struct expressionNode *expression_node)
void executeBinaryExpression(struct expressionNode *expression_node)
void executeUnaryExpression(struct expressionNode *expression_node)
void executeConstructNode(struct constructNode *construct_node)
void executeIfConstruct(struct ifConstruct *if_construct)
int executeIfBlock(struct ifBlock* if_block)
int executeElifBlock(struct elifBlock* elif_block)
void executeElseBlock(struct elseBlock* else_block)
void executeForConstruct(struct forConstruct *for_construct)
void executeWhileConstruct(struct whileConstruct *while_construct)
```



```
///////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                               //
// ------------------------------------ Node Helper Functions ------------------------------------ //
//                                                                                               //
///////////////////////////////////////////////////////////////////////////////////////////////////

char* toString(struct valueNode* value_node)
struct variable* getVariable(char* name)
struct valueNode* getDefaultValueNode()
union varvalue* cloneValue(char* type, union varvalue* value)
struct variable* insert(char* symname, char* type, union varvalue *val)
void removeVariables(int scope)
void swap(char* name1, char* name2)
void display()
void switchUnionValue(union varvalue* value, char* fromType, char* toType)
struct statementSetNode* reverseStatementSetNodes(struct statementSetNode* root)
```

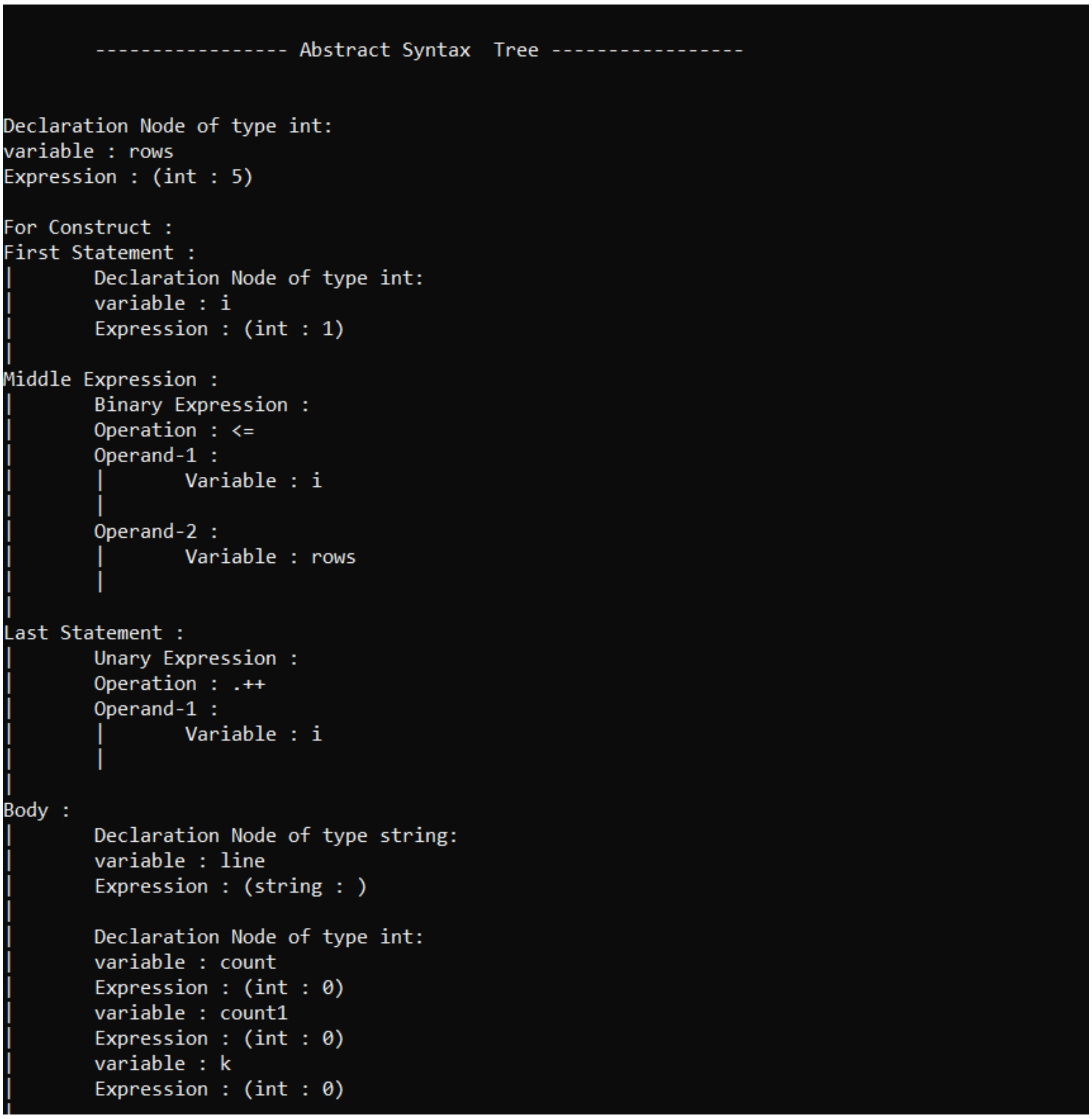
Example of generated Abstract Syntax Tree

Following is the sample but complex long program. We will try to generate the AST for it.

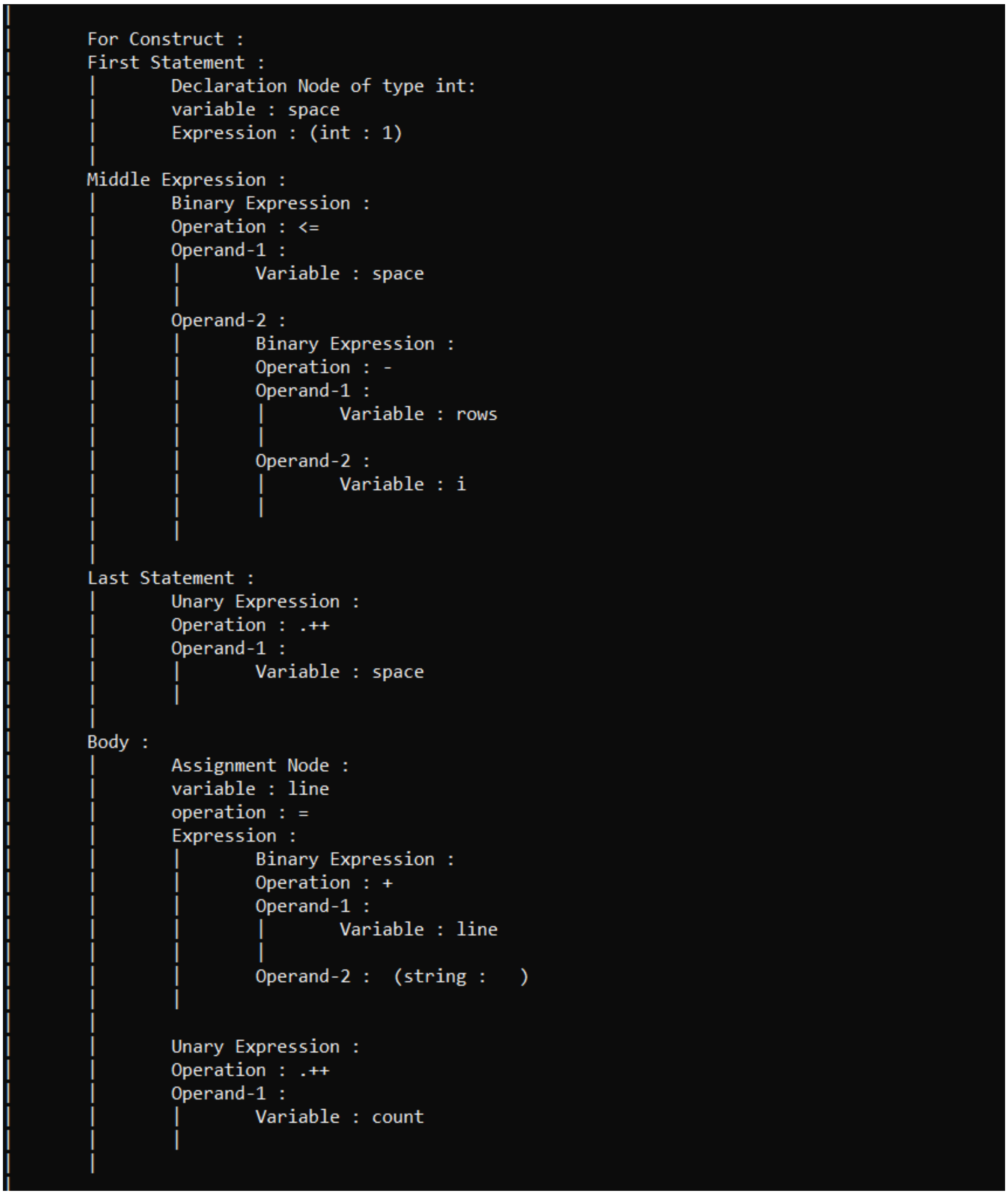


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main() {

        int rows = 5

        for(int i = 1; i <= rows; i++) {
                string line = ""
                int count = 0, count1 = 0, k = 0

                for(int space = 1; space <= rows - i; space++) {
                        line = line + "  "
                        count++
                }

                while(k != 2 * i - 1) {
                        if(count <= rows - 1) {
                                line = line + (i + k) + " "
                                count++
                        } else {
                                count1++
                                line = line + (i + k - 2 * count1) + " "
                        }
                        k++
                }

                print(line)
        }
}
---------------------------------
```

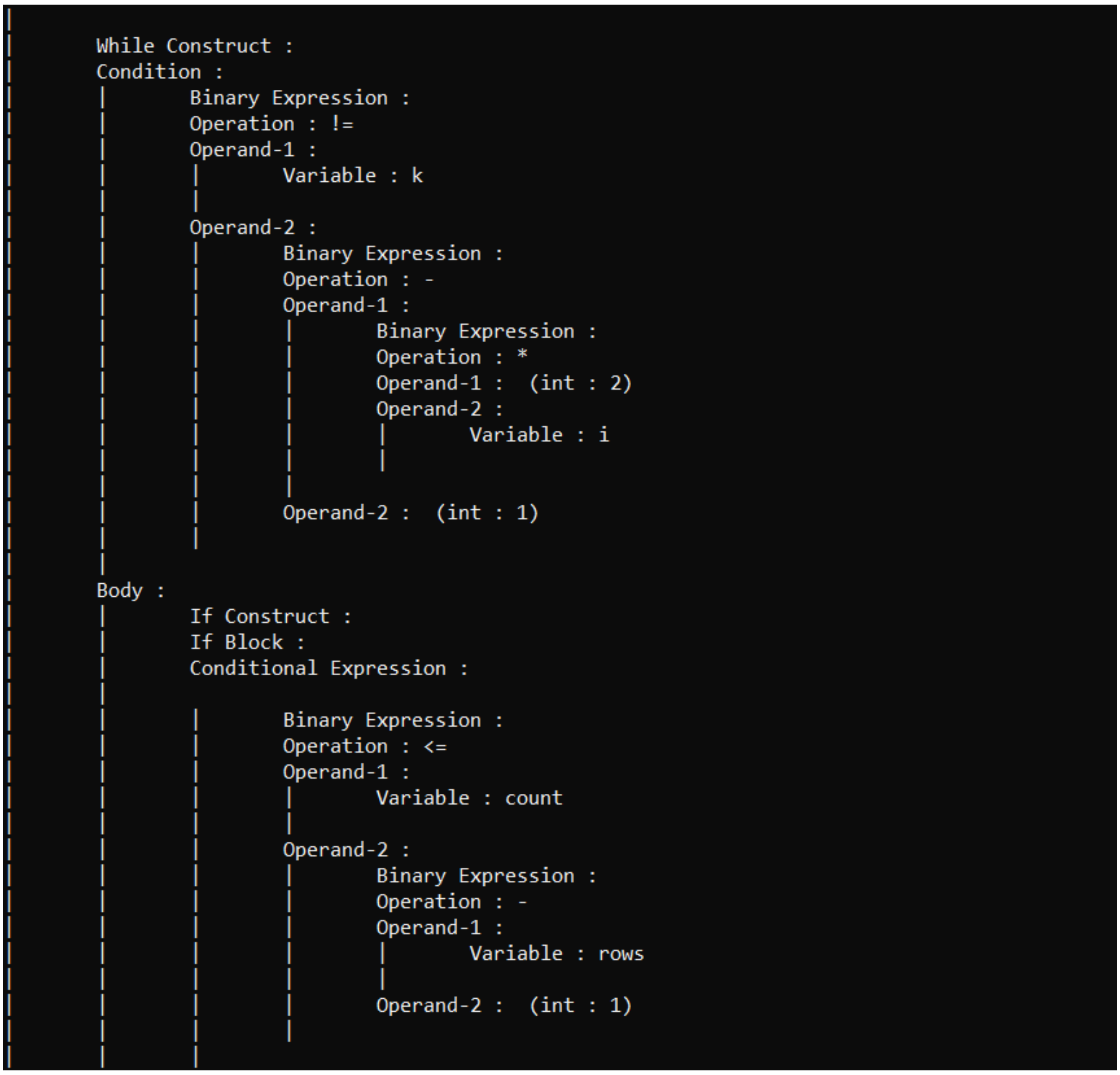
Following are the screen snips of Abstract Syntax Tree generated for the above program. (Which later is executed by node execution functions in an appropriate flow).



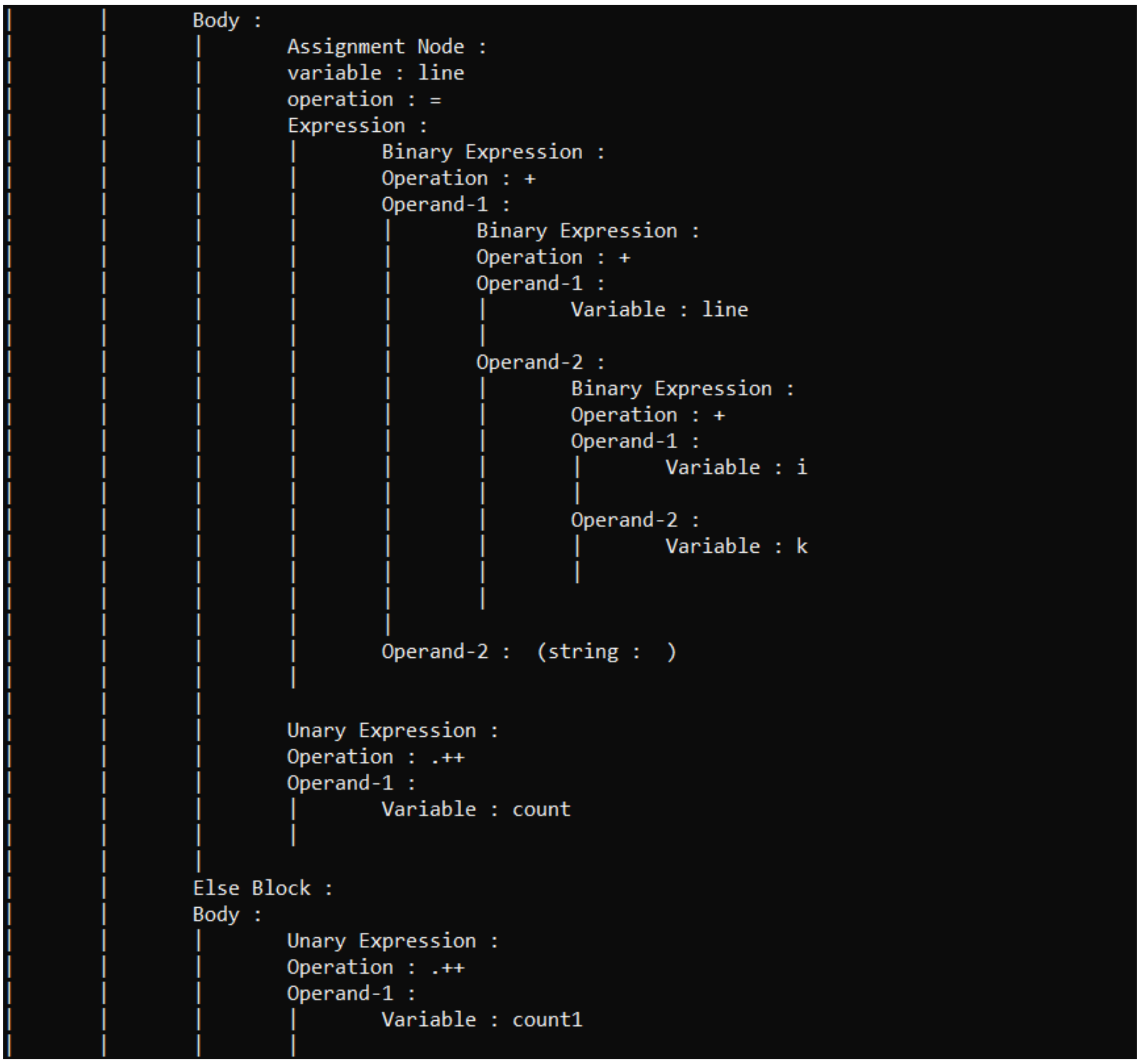
```
        ---------------- Abstract Syntax  Tree -----------------


Declaration Node of type int:
variable : rows
Expression : (int : 5)

For Construct :
First Statement :
|       Declaration Node of type int:
|       variable : i
|       Expression : (int : 1)
|
Middle Expression :
|       Binary Expression :
|       Operation : <=
|       Operand-1 :
|       |       Variable : i
|       |
|       Operand-2 :
|       |       Variable : rows
|       |
|
Last Statement :
|       Unary Expression :
|       Operation : .++
|       Operand-1 :
|       |       Variable : i
|       |
|
Body :
|       Declaration Node of type string:
|       variable : line
|       Expression : (string : )
|
|       Declaration Node of type int:
|       variable : count
|       Expression : (int : 0)
|       variable : count1
|       Expression : (int : 0)
|       variable : k
|       Expression : (int : 0)
|
```



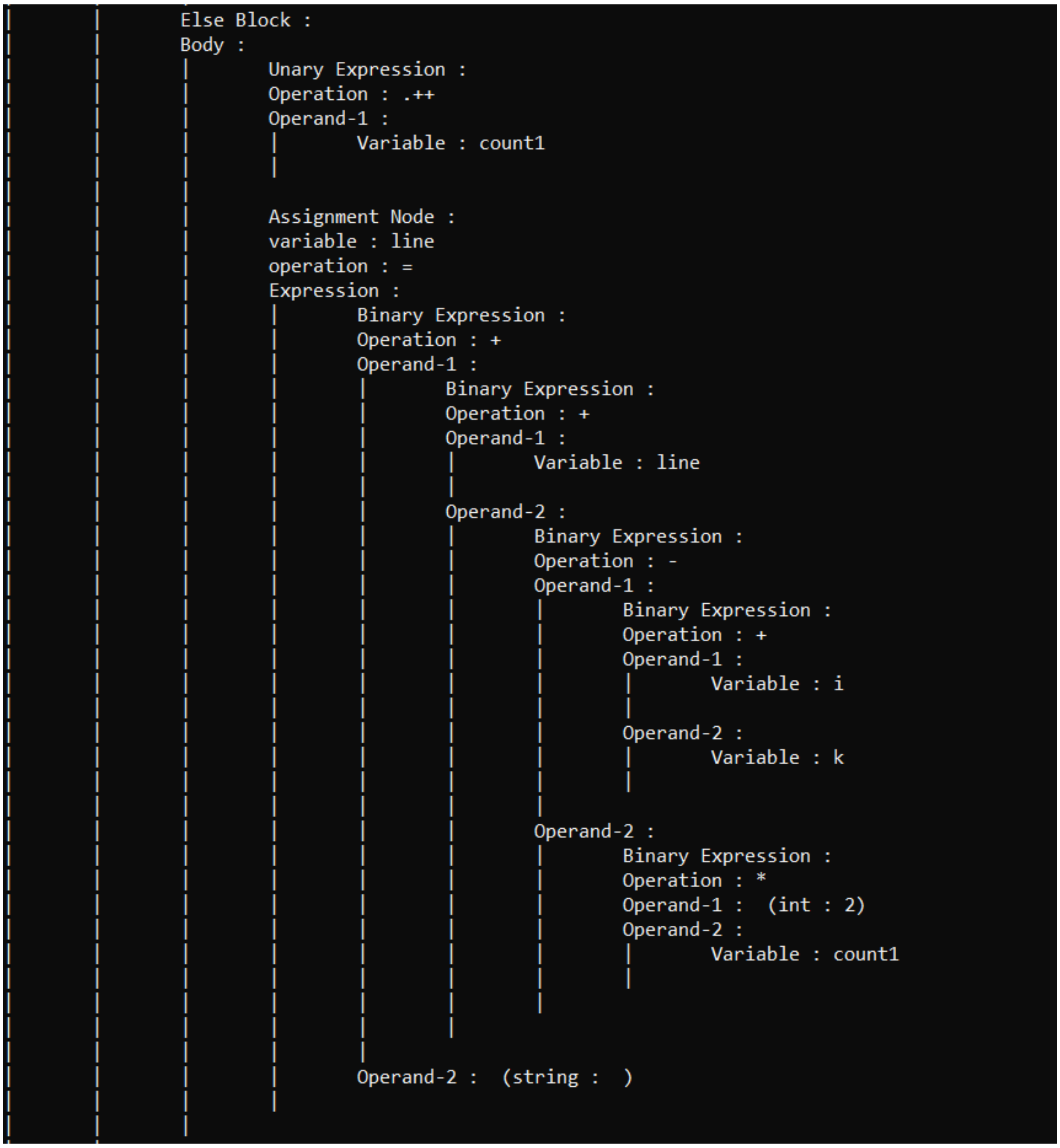
```
|
|       For Construct :
|       First Statement :
|       |       Declaration Node of type int:
|       |       variable : space
|       |       Expression : (int : 1)
|       |
|       Middle Expression :
|       |       Binary Expression :
|       |       Operation : <=
|       |       Operand-1 :
|       |       |       Variable : space
|       |       |
|       |       Operand-2 :
|       |       |       Binary Expression :
|       |       |       Operation : -
|       |       |       Operand-1 :
|       |       |       |       Variable : rows
|       |       |       |
|       |       |       Operand-2 :
|       |       |       |       Variable : i
|       |       |       |
|       |       |
|       |
|       Last Statement :
|       |       Unary Expression :
|       |       Operation : .++
|       |       Operand-1 :
|       |       |       Variable : space
|       |       |
|       |
|       Body :
|       |       Assignment Node :
|       |       variable : line
|       |       operation : =
|       |       Expression :
|       |       |       Binary Expression :
|       |       |       Operation : +
|       |       |       Operand-1 :
|       |       |       |       Variable : line
|       |       |       |
|       |       |       Operand-2 :  (string :   )
|       |       |
|       |
|       |       Unary Expression :
|       |       Operation : .++
|       |       Operand-1 :
|       |       |       Variable : count
|       |       |
|       |
```



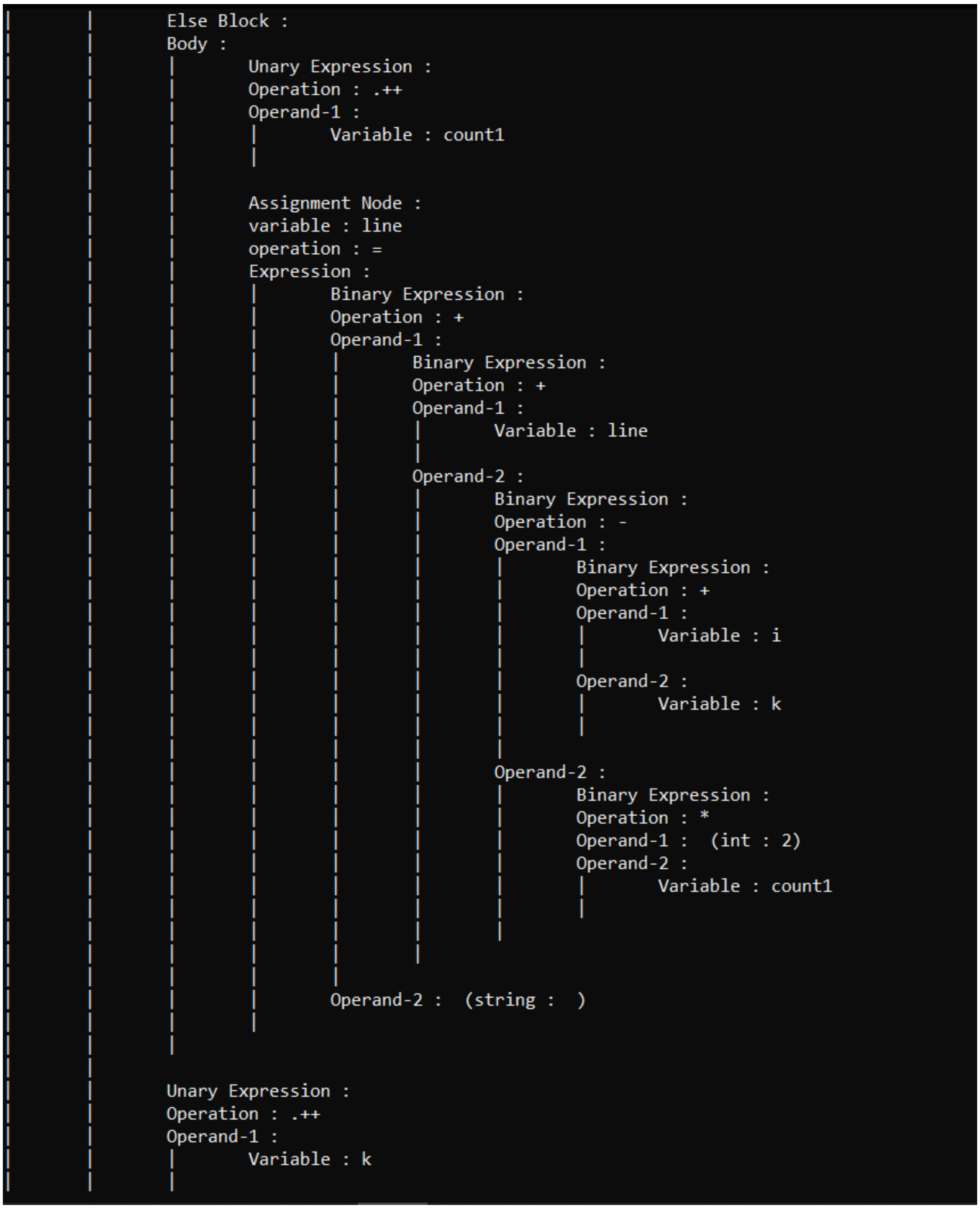
```
|
|       While Construct :
|       Condition :
|       |       Binary Expression :
|       |       Operation : !=
|       |       Operand-1 :
|       |       |       Variable : k
|       |       |
|       |       Operand-2 :
|       |       |       Binary Expression :
|       |       |       Operation : -
|       |       |       Operand-1 :
|       |       |       |       Binary Expression :
|       |       |       |       Operation : *
|       |       |       |       Operand-1 :  (int : 2)
|       |       |       |       Operand-2 :
|       |       |       |       |       Variable : i
|       |       |       |       |
|       |       |       |
|       |       |       Operand-2 :  (int : 1)
|       |       |
|       |
|       Body :
|       |       If Construct :
|       |       If Block :
|       |       Conditional Expression :
|       |
|       |       |       Binary Expression :
|       |       |       Operation : <=
|       |       |       Operand-1 :
|       |       |       |       Variable : count
|       |       |       |
|       |       |       Operand-2 :
|       |       |       |       Binary Expression :
|       |       |       |       Operation : -
|       |       |       |       Operand-1 :
|       |       |       |       |       Variable : rows
|       |       |       |       |
|       |       |       |       Operand-2 :  (int : 1)
|       |       |       |
|       |       |
```



```
|       |       Body :
|       |       |       Assignment Node :
|       |       |       variable : line
|       |       |       operation : =
|       |       |       Expression :
|       |       |       |       Binary Expression :
|       |       |       |       Operation : +
|       |       |       |       Operand-1 :
|       |       |       |       |       Binary Expression :
|       |       |       |       |       Operation : +
|       |       |       |       |       Operand-1 :
|       |       |       |       |       |       Variable : line
|       |       |       |       |       |
|       |       |       |       |       Operand-2 :
|       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       Operation : +
|       |       |       |       |       |       Operand-1 :
|       |       |       |       |       |       |       Variable : i
|       |       |       |       |       |       |
|       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       Variable : k
|       |       |       |       |       |       |
|       |       |       |       |       |
|       |       |       |       |
|       |       |       |       Operand-2 :  (string :  )
|       |       |       |
|       |       |
|       |       |       Unary Expression :
|       |       |       Operation : .++
|       |       |       Operand-1 :
|       |       |       |       Variable : count
|       |       |       |
|       |       |
|       |       Else Block :
|       |       Body :
|       |       |       Unary Expression :
|       |       |       Operation : .++
|       |       |       Operand-1 :
|       |       |       |       Variable : count1
|       |       |       |
```



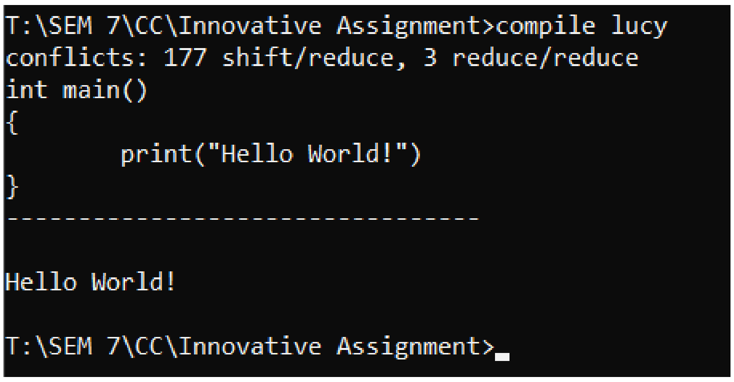
```
|       |       Else Block :
|       |       Body :
|       |       |       Unary Expression :
|       |       |       Operation : .++
|       |       |       Operand-1 :
|       |       |       |       Variable : count1
|       |       |       |
|       |       |
|       |       |       Assignment Node :
|       |       |       variable : line
|       |       |       operation : =
|       |       |       Expression :
|       |       |       |       Binary Expression :
|       |       |       |       Operation : +
|       |       |       |       Operand-1 :
|       |       |       |       |       Binary Expression :
|       |       |       |       |       Operation : +
|       |       |       |       |       Operand-1 :
|       |       |       |       |       |       Variable : line
|       |       |       |       |       |
|       |       |       |       |       Operand-2 :
|       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       Operation : -
|       |       |       |       |       |       Operand-1 :
|       |       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       |       Operation : +
|       |       |       |       |       |       |       Operand-1 :
|       |       |       |       |       |       |       |       Variable : i
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       |       Variable : k
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |
|       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       |       Operation : *
|       |       |       |       |       |       |       Operand-1 :  (int : 2)
|       |       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       |       Variable : count1
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |
|       |       |       |       |       |
|       |       |       |       |
|       |       |       |       Operand-2 :  (string :  )
|       |       |       |
|       |       |
```



```
|       |       Else Block :
|       |       Body :
|       |       |       Unary Expression :
|       |       |       Operation : .++
|       |       |       Operand-1 :
|       |       |       |       Variable : count1
|       |       |       |
|       |       |
|       |       |       Assignment Node :
|       |       |       variable : line
|       |       |       operation : =
|       |       |       Expression :
|       |       |       |       Binary Expression :
|       |       |       |       Operation : +
|       |       |       |       Operand-1 :
|       |       |       |       |       Binary Expression :
|       |       |       |       |       Operation : +
|       |       |       |       |       Operand-1 :
|       |       |       |       |       |       Variable : line
|       |       |       |       |       |
|       |       |       |       |       Operand-2 :
|       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       Operation : -
|       |       |       |       |       |       Operand-1 :
|       |       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       |       Operation : +
|       |       |       |       |       |       |       Operand-1 :
|       |       |       |       |       |       |       |       Variable : i
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       |       Variable : k
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |
|       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       Binary Expression :
|       |       |       |       |       |       |       Operation : *
|       |       |       |       |       |       |       Operand-1 :  (int : 2)
|       |       |       |       |       |       |       Operand-2 :
|       |       |       |       |       |       |       |       Variable : count1
|       |       |       |       |       |       |       |
|       |       |       |       |       |       |
|       |       |       |       |       |
|       |       |       |       |
|       |       |       |       Operand-2 :  (string :  )
|       |       |       |
|       |       |
|       |
|       |       Unary Expression :
|       |       Operation : .++
|       |       Operand-1 :
|       |       |       Variable : k
|       |       |
```

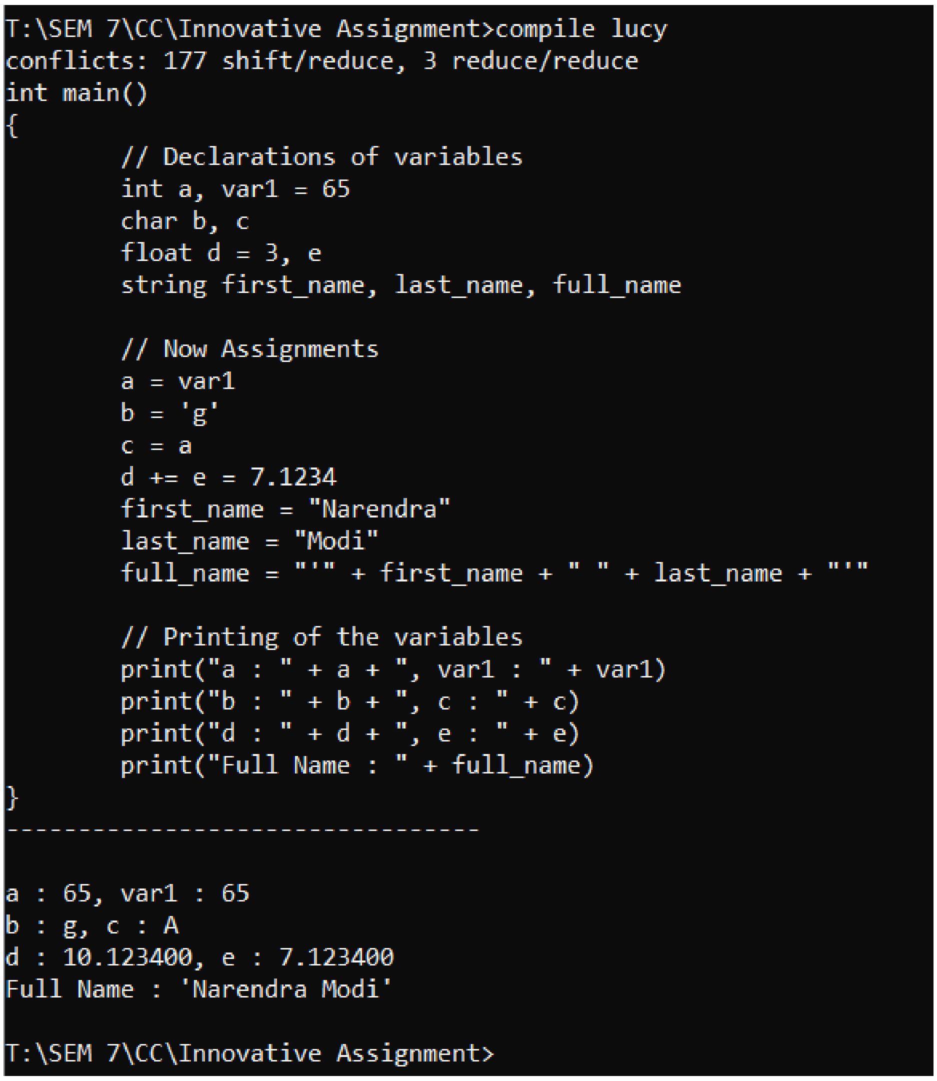
Sample Code Executions

Hello World Program

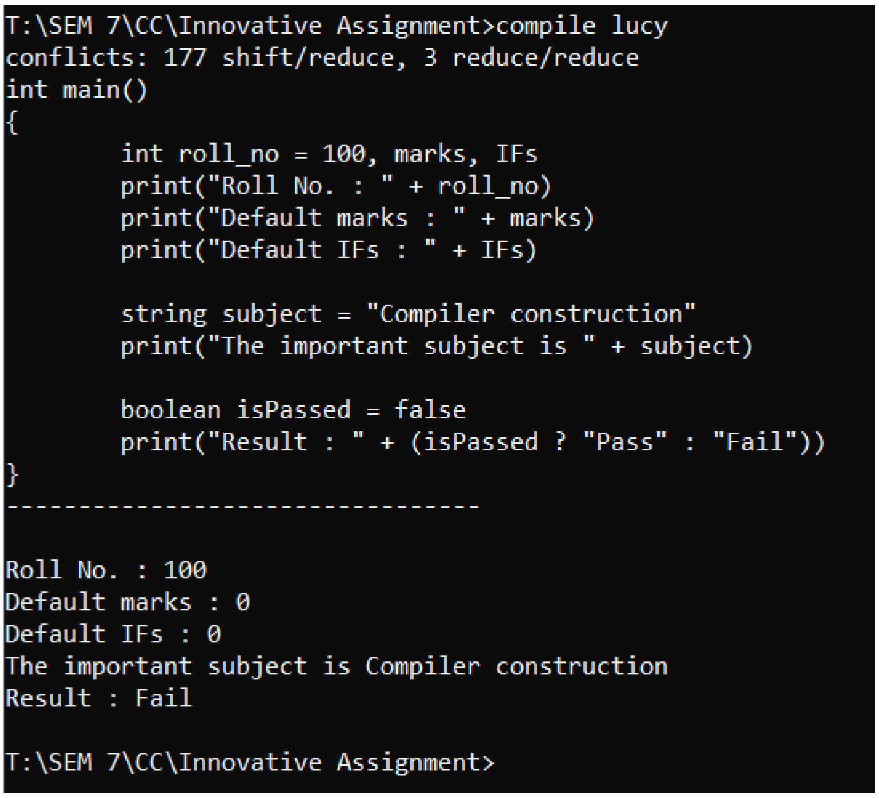


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        print("Hello World!")
}
---------------------------------

Hello World!

T:\SEM 7\CC\Innovative Assignment>
```

Declaration and Assignment Statements

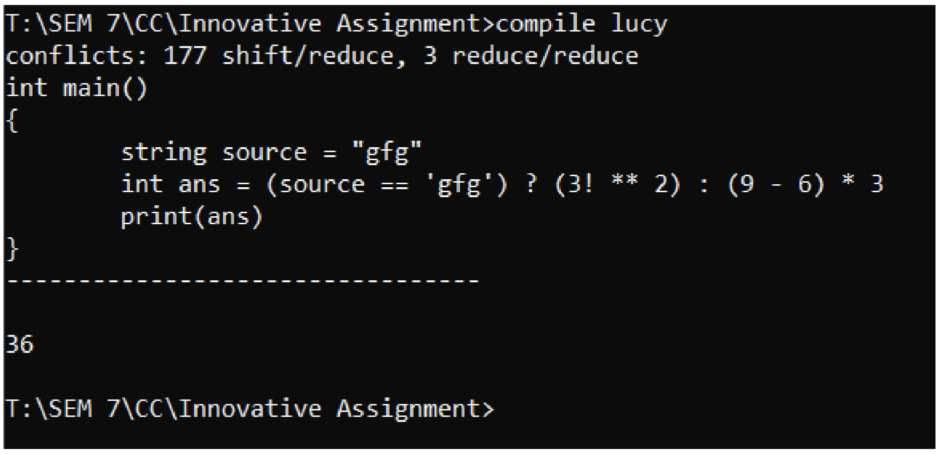


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        // Declarations of variables
        int a, var1 = 65
        char b, c
        float d = 3, e
        string first_name, last_name, full_name

        // Now Assignments
        a = var1
        b = 'g'
        c = a
        d += e = 7.1234
        first_name = "Narendra"
        last_name = "Modi"
        full_name = "'" + first_name + " " + last_name + "'"

        // Printing of the variables
        print("a : " + a + ", var1 : " + var1)
        print("b : " + b + ", c : " + c)
        print("d : " + d + ", e : " + e)
        print("Full Name : " + full_name)
}
---------------------------------

a : 65, var1 : 65
b : g, c : A
d : 10.123400, e : 7.123400
Full Name : 'Narendra Modi'

T:\SEM 7\CC\Innovative Assignment>
```

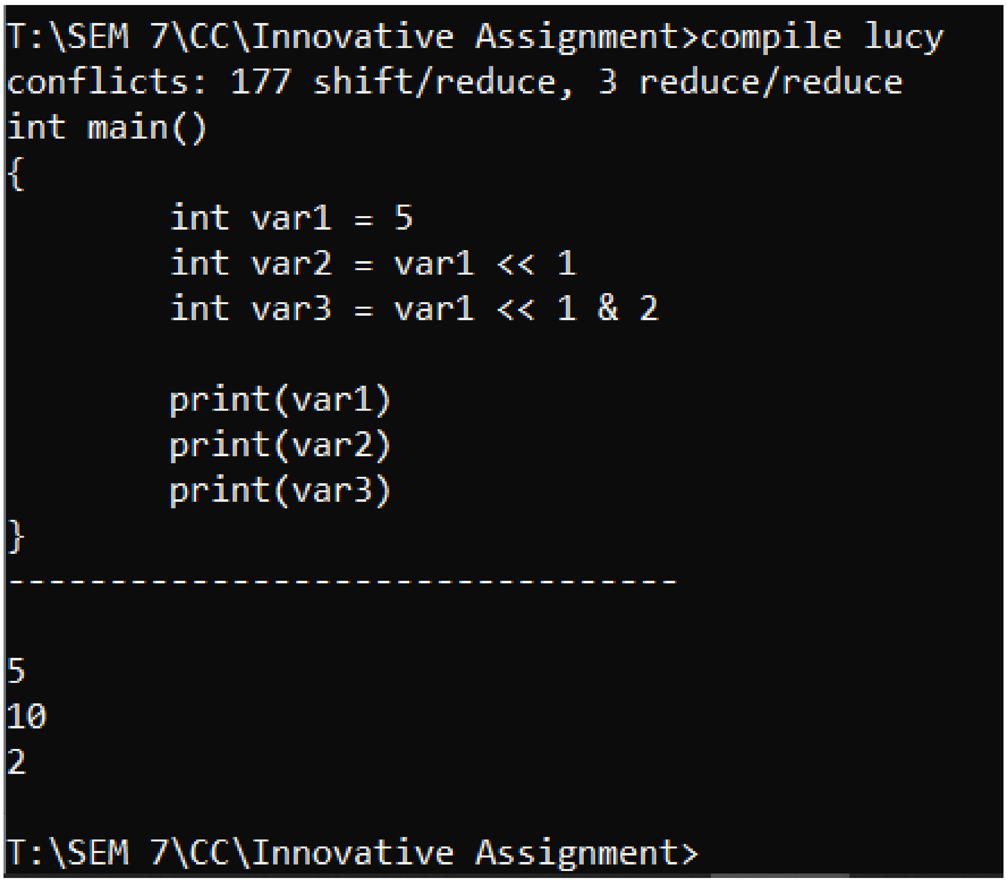


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int roll_no = 100, marks, IFs
        print("Roll No. : " + roll_no)
        print("Default marks : " + marks)
        print("Default IFs : " + IFs)

        string subject = "Compiler construction"
        print("The important subject is " + subject)

        boolean isPassed = false
        print("Result : " + (isPassed ? "Pass" : "Fail"))
}
---------------------------------

Roll No. : 100
Default marks : 0
Default IFs : 0
The important subject is Compiler construction
Result : Fail

T:\SEM 7\CC\Innovative Assignment>
```

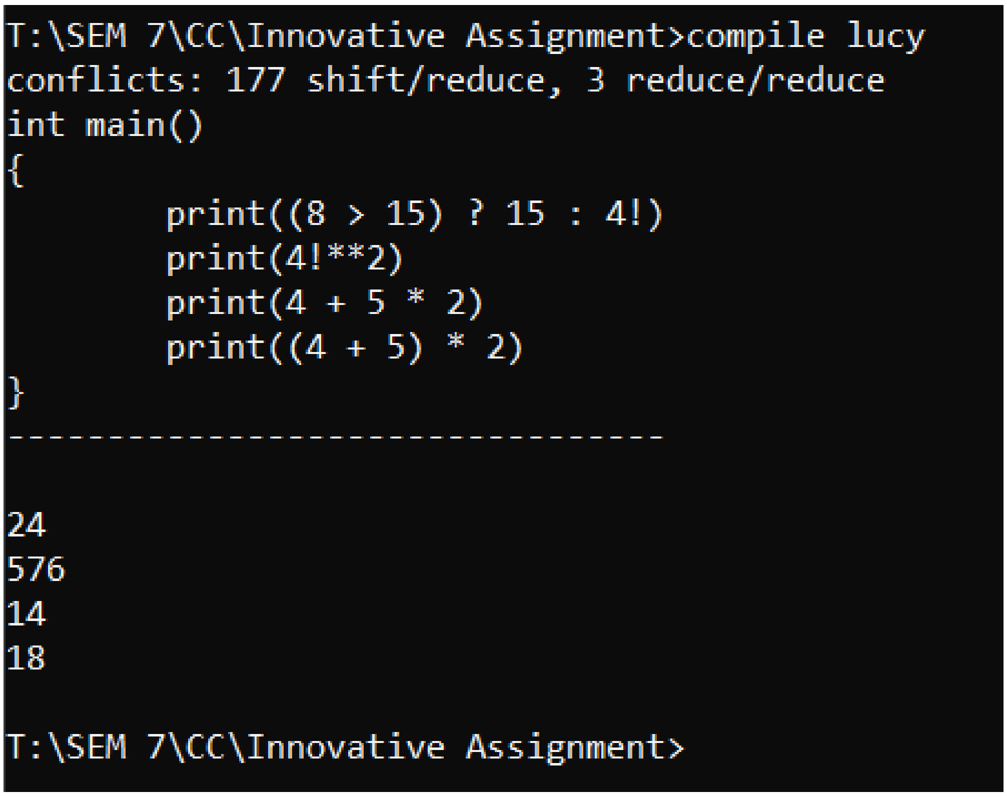
Expressions



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        string source = "gfg"
        int ans = (source == 'gfg') ? (3! ** 2) : (9 - 6) * 3
        print(ans)
}
---------------------------------

36

T:\SEM 7\CC\Innovative Assignment>
```

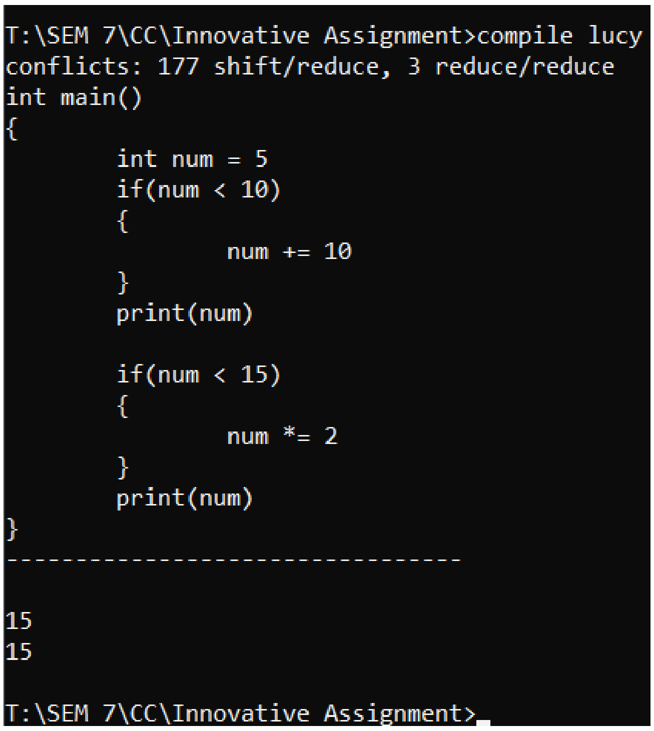


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int var1 = 5
        int var2 = var1 << 1
        int var3 = var1 << 1 & 2

        print(var1)
        print(var2)
        print(var3)
}
---------------------------------


5
10
2

T:\SEM 7\CC\Innovative Assignment>
```

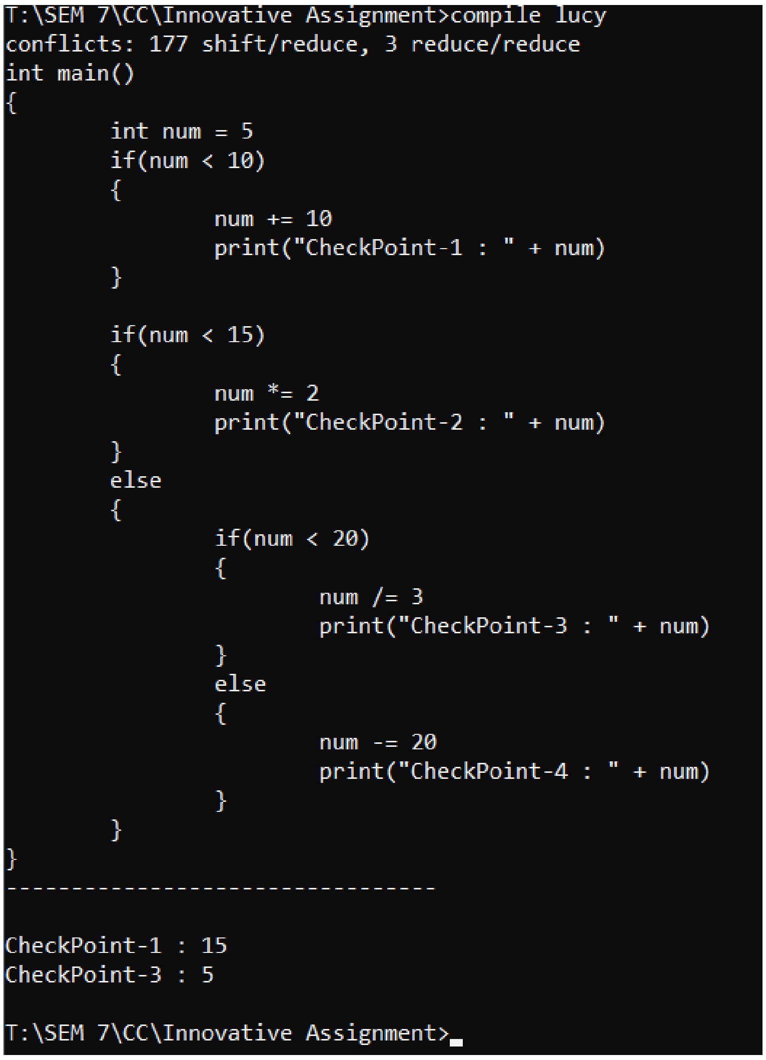


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        print((8 > 15) ? 15 : 4!)
        print(4!**2)
        print(4 + 5 * 2)
        print((4 + 5) * 2)
}
---------------------------------


24
576
14
18

T:\SEM 7\CC\Innovative Assignment>
```

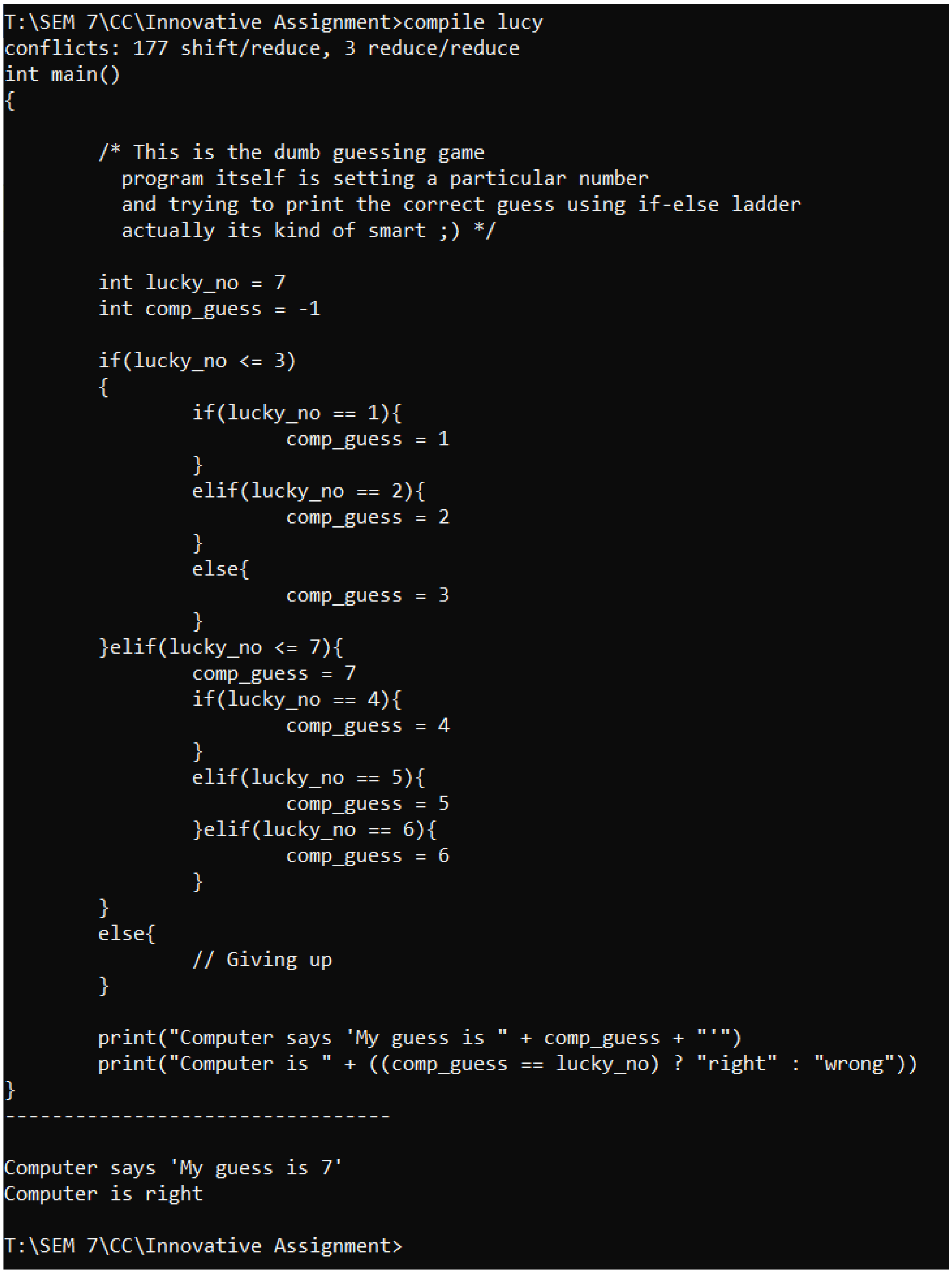
If Construct



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int num = 5
        if(num < 10)
        {
                num += 10
        }
        print(num)

        if(num < 15)
        {
                num *= 2
        }
        print(num)
}
---------------------------------

15
15

T:\SEM 7\CC\Innovative Assignment>
```

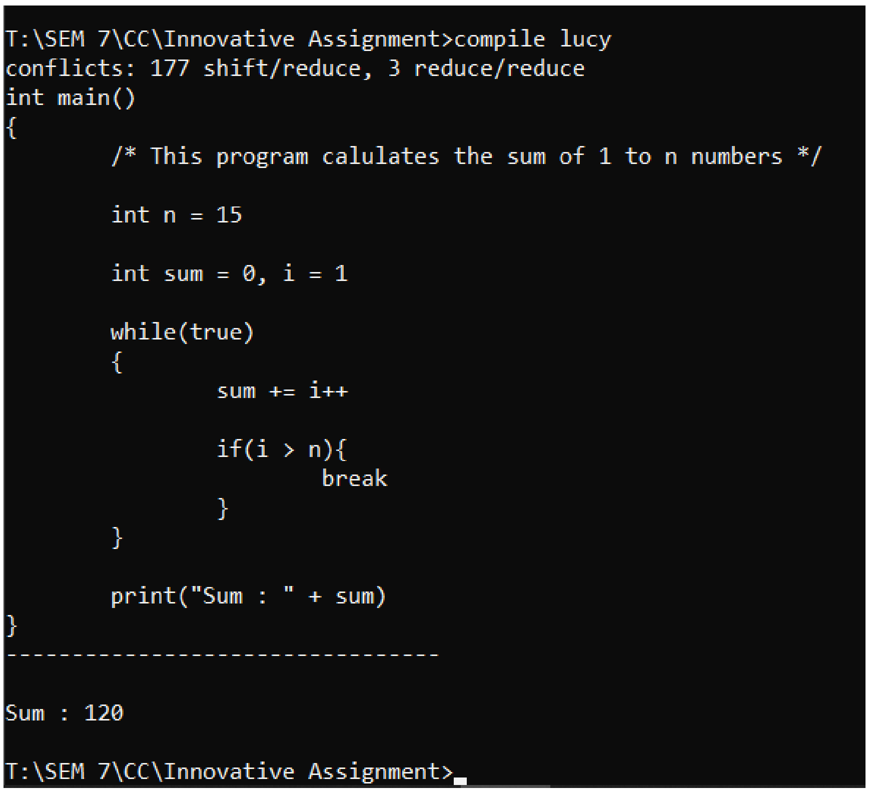


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int num = 5
        if(num < 10)
        {
                num += 10
                print("CheckPoint-1 : " + num)
        }

        if(num < 15)
        {
                num *= 2
                print("CheckPoint-2 : " + num)
        }
        else
        {
                if(num < 20)
                {
                        num /= 3
                        print("CheckPoint-3 : " + num)
                }
                else
                {
                        num -= 20
                        print("CheckPoint-4 : " + num)
                }
        }
}
---------------------------------

CheckPoint-1 : 15
CheckPoint-3 : 5

T:\SEM 7\CC\Innovative Assignment>
```



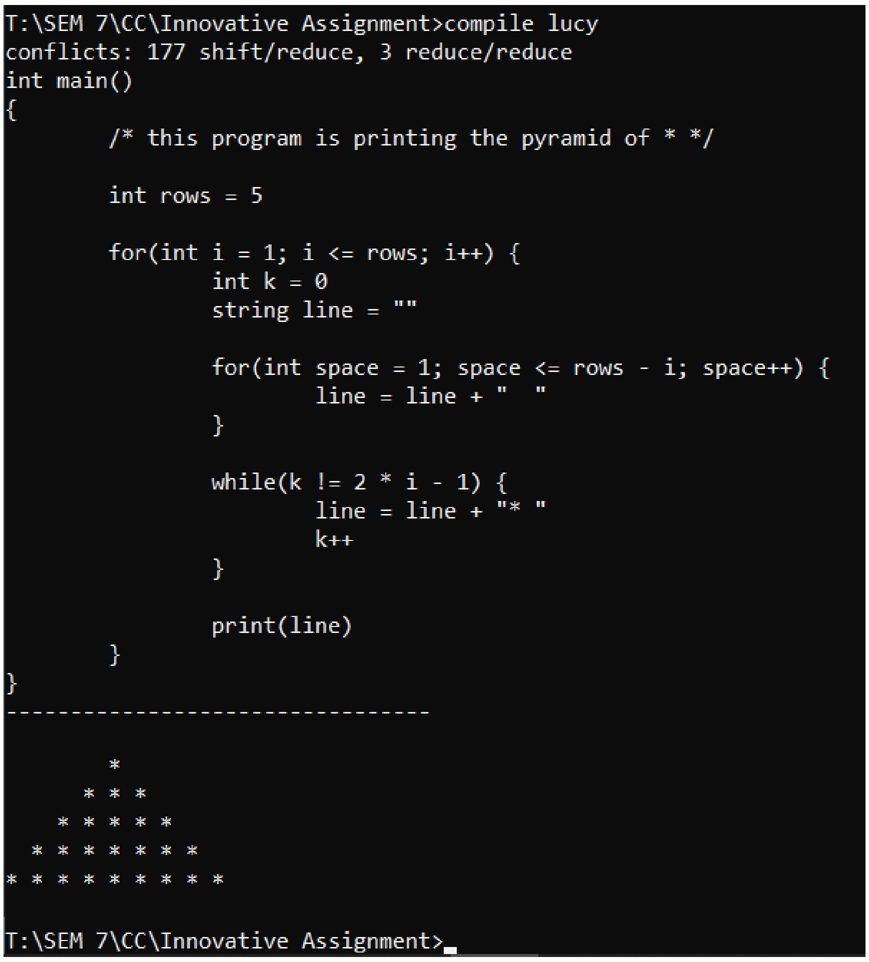
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{

        /* This is the dumb guessing game
          program itself is setting a particular number
          and trying to print the correct guess using if-else ladder
          actually its kind of smart ;) */

        int lucky_no = 7
        int comp_guess = -1

        if(lucky_no <= 3)
        {
                if(lucky_no == 1){
                        comp_guess = 1
                }
                elif(lucky_no == 2){
                        comp_guess = 2
                }
                else{
                        comp_guess = 3
                }
        }elif(lucky_no <= 7){
                comp_guess = 7
                if(lucky_no == 4){
                        comp_guess = 4
                }
                elif(lucky_no == 5){
                        comp_guess = 5
                }elif(lucky_no == 6){
                        comp_guess = 6
                }
        }
        else{
                // Giving up
        }

        print("Computer says 'My guess is " + comp_guess + "'")
        print("Computer is " + ((comp_guess == lucky_no) ? "right" : "wrong"))
}
---------------------------------

Computer says 'My guess is 7'
Computer is right

T:\SEM 7\CC\Innovative Assignment>
```

While Construct

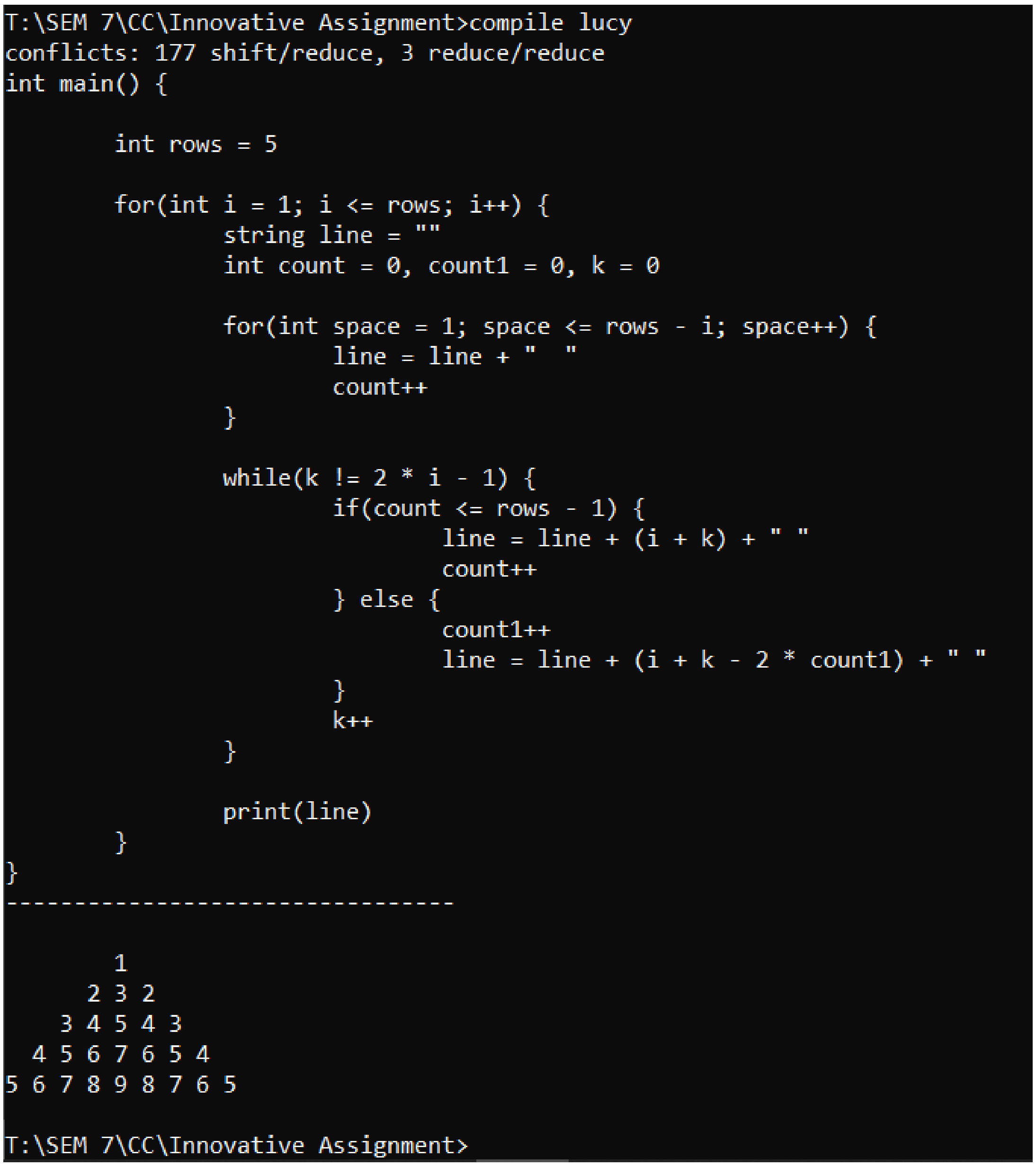


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        /* This program calulates the sum of 1 to n numbers */

        int n = 15

        int sum = 0, i = 1

        while(true)
        {
                sum += i++

                if(i > n){
                        break
                }
        }

        print("Sum : " + sum)
}
---------------------------------

Sum : 120

T:\SEM 7\CC\Innovative Assignment>
```

For Construct

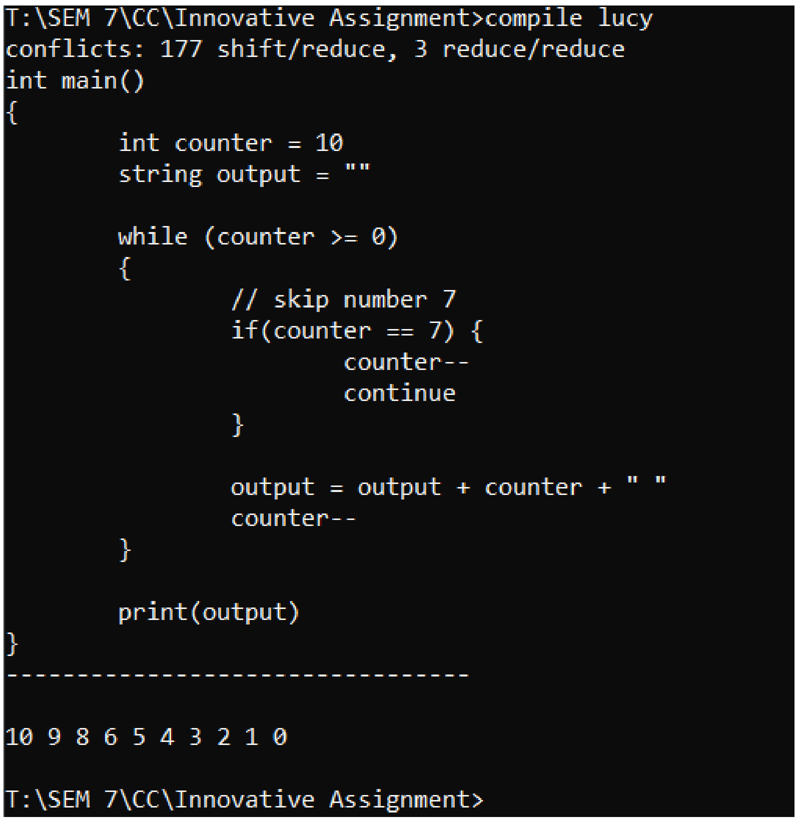


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        /* this program is printing the pyramid of * */

        int rows = 5

        for(int i = 1; i <= rows; i++) {
                int k = 0
                string line = ""

                for(int space = 1; space <= rows - i; space++) {
                        line = line + "  "
                }

                while(k != 2 * i - 1) {
                        line = line + "* "
                        k++
                }

                print(line)
        }
}
---------------------------------

        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *

T:\SEM 7\CC\Innovative Assignment>
```



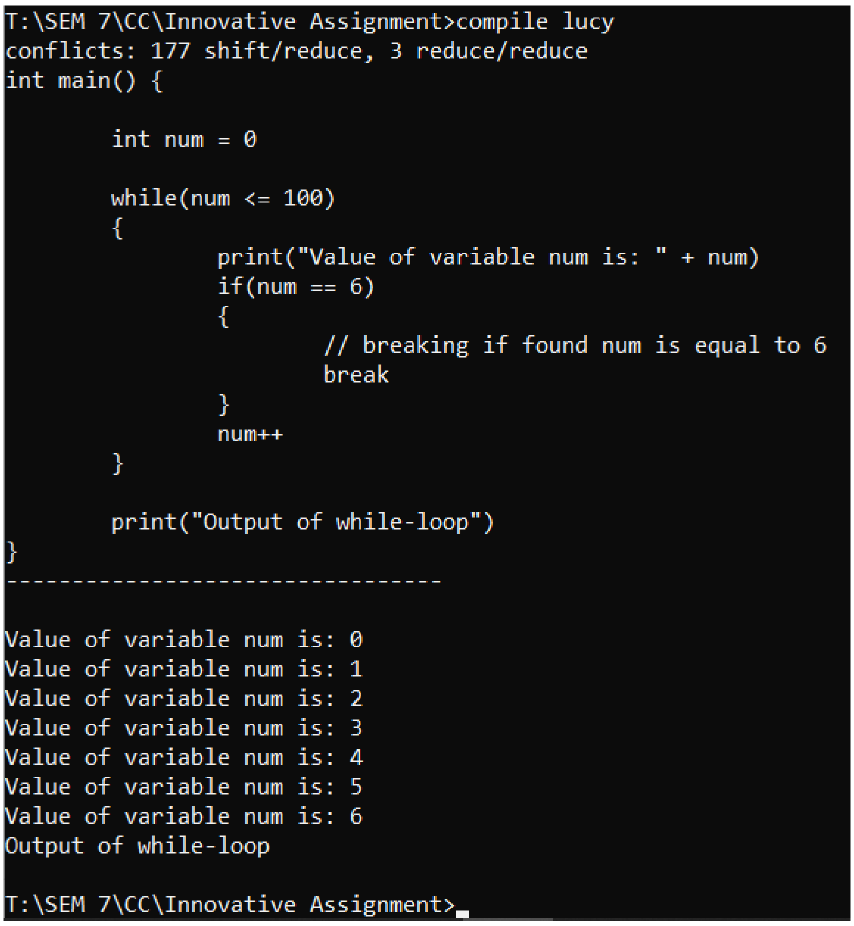
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main() {

        int rows = 5

        for(int i = 1; i <= rows; i++) {
                string line = ""
                int count = 0, count1 = 0, k = 0

                for(int space = 1; space <= rows - i; space++) {
                        line = line + "  "
                        count++
                }

                while(k != 2 * i - 1) {
                        if(count <= rows - 1) {
                                line = line + (i + k) + " "
                                count++
                        } else {
                                count1++
                                line = line + (i + k - 2 * count1) + " "
                        }
                        k++
                }

                print(line)
        }
}
---------------------------------


        1
      2 3 2
    3 4 5 4 3
  4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5

T:\SEM 7\CC\Innovative Assignment>
```

Continue Statement



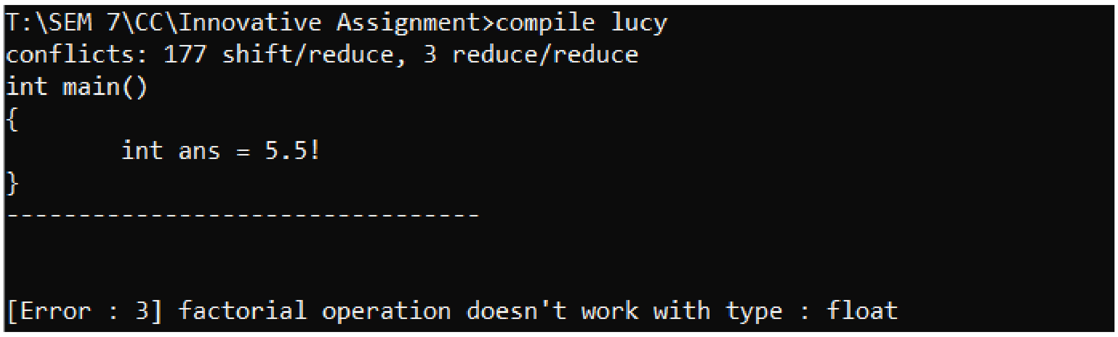
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int counter = 10
        string output = ""

        while (counter >= 0)
        {
                // skip number 7
                if(counter == 7) {
                        counter--
                        continue
                }

                output = output + counter + " "
                counter--
        }

        print(output)
}
--------------------------------

10 9 8 6 5 4 3 2 1 0

T:\SEM 7\CC\Innovative Assignment>
```

Break Statement

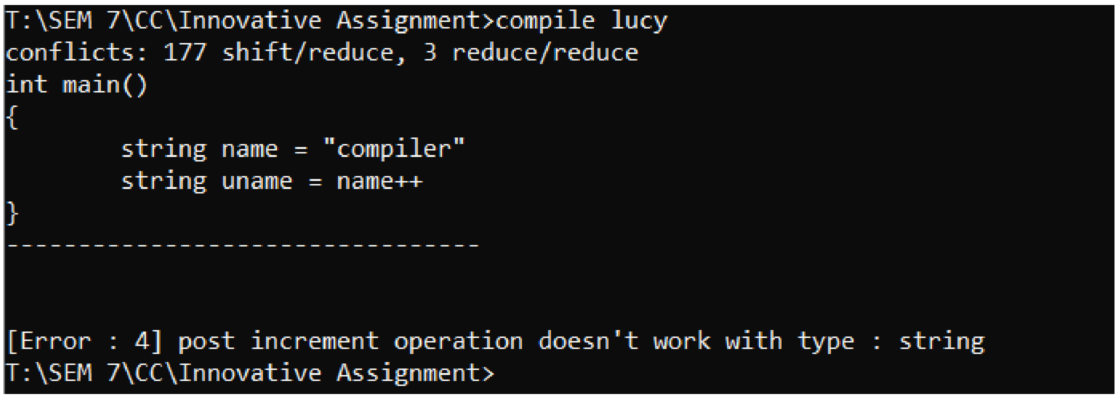


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main() {

        int num = 0

        while(num <= 100)
        {
                print("Value of variable num is: " + num)
                if(num == 6)
                {
                        // breaking if found num is equal to 6
                        break
                }
                num++
        }

        print("Output of while-loop")
}
--------------------------------

Value of variable num is: 0
Value of variable num is: 1
Value of variable num is: 2
Value of variable num is: 3
Value of variable num is: 4
Value of variable num is: 5
Value of variable num is: 6
Output of while-loop

T:\SEM 7\CC\Innovative Assignment>
```

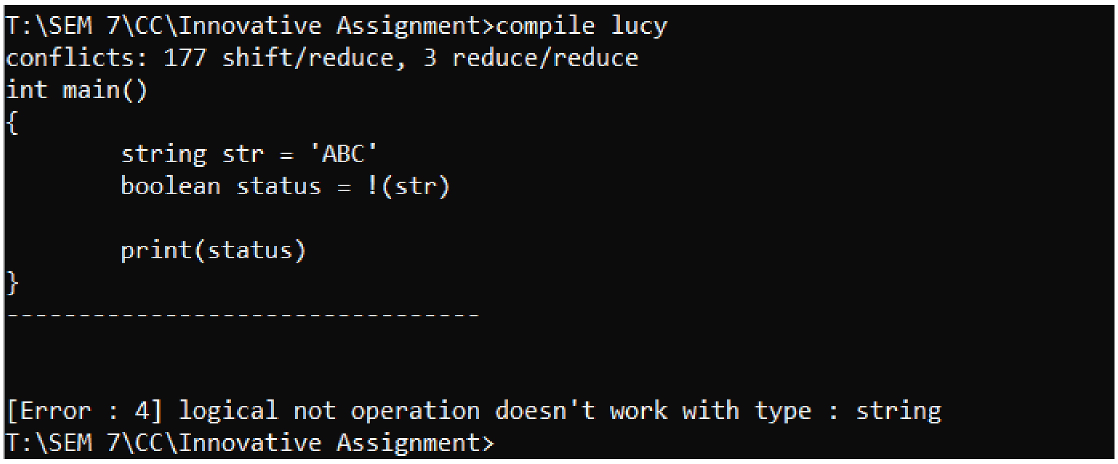
Some Semantic Errors



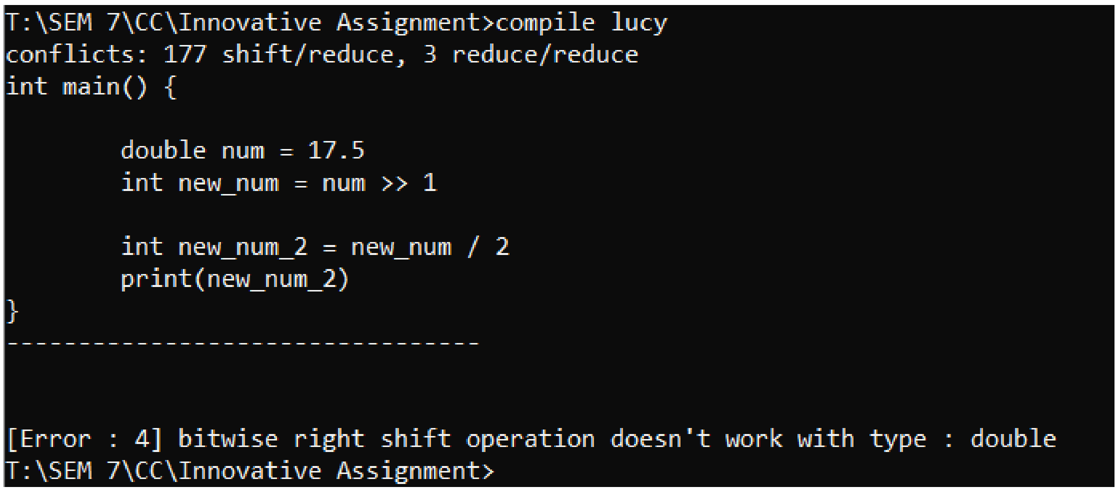
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int ans = 5.5!
}
---------------------------------


[Error : 3] factorial operation doesn't work with type : float
```



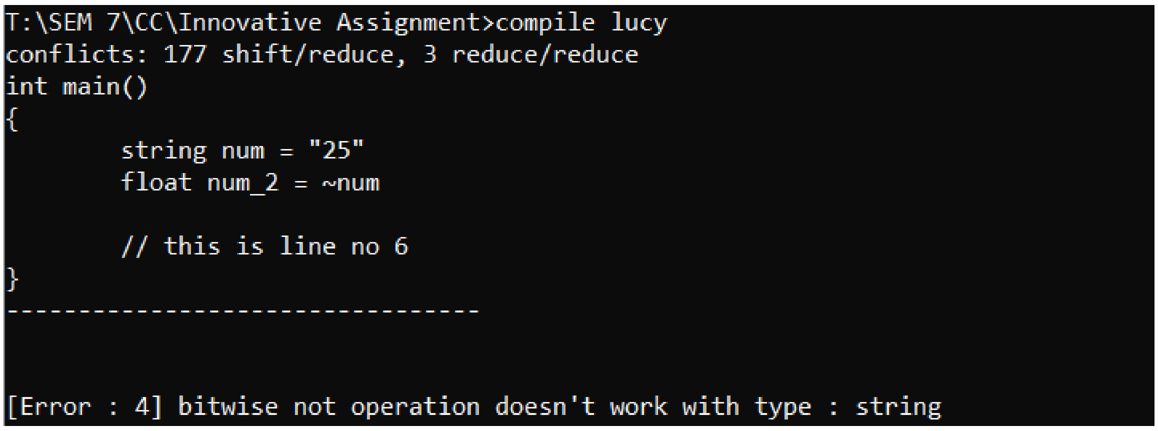
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        string name = "compiler"
        string uname = name++
}
---------------------------------


[Error : 4] post increment operation doesn't work with type : string
T:\SEM 7\CC\Innovative Assignment>
```



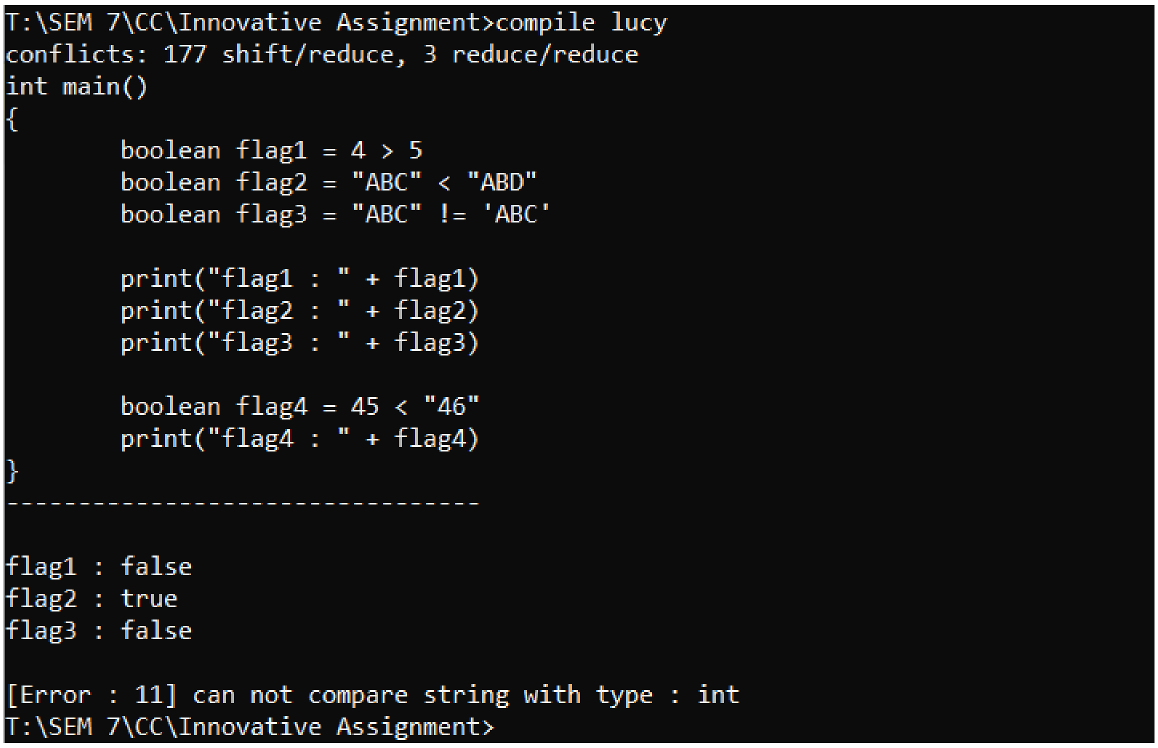
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        string str = 'ABC'
        boolean status = !(str)

        print(status)
}
---------------------------------


[Error : 4] logical not operation doesn't work with type : string
T:\SEM 7\CC\Innovative Assignment>
```



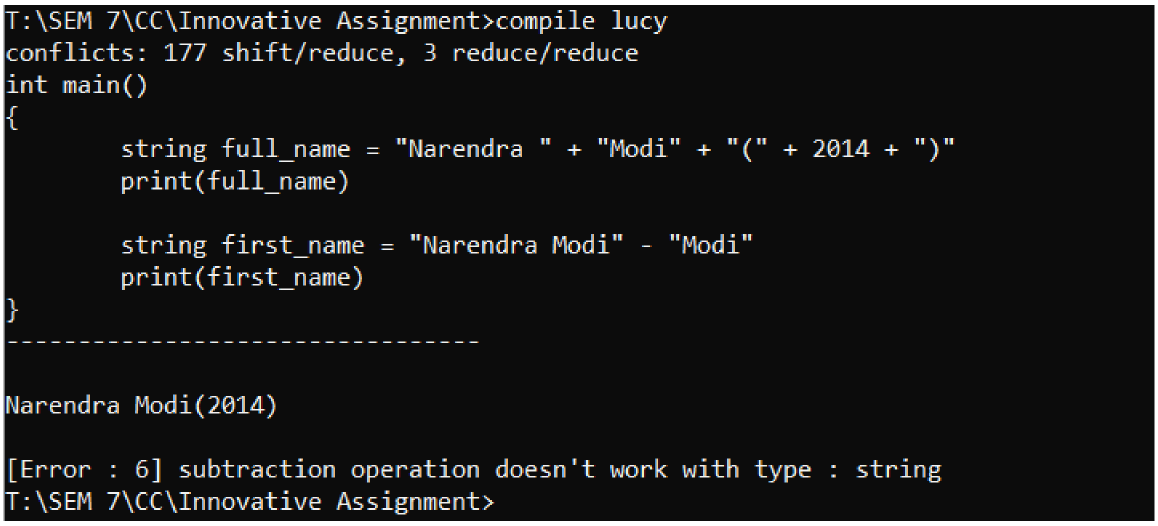
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main() {

        double num = 17.5
        int new_num = num >> 1

        int new_num_2 = new_num / 2
        print(new_num_2)
}
---------------------------------


[Error : 4] bitwise right shift operation doesn't work with type : double
T:\SEM 7\CC\Innovative Assignment>
```



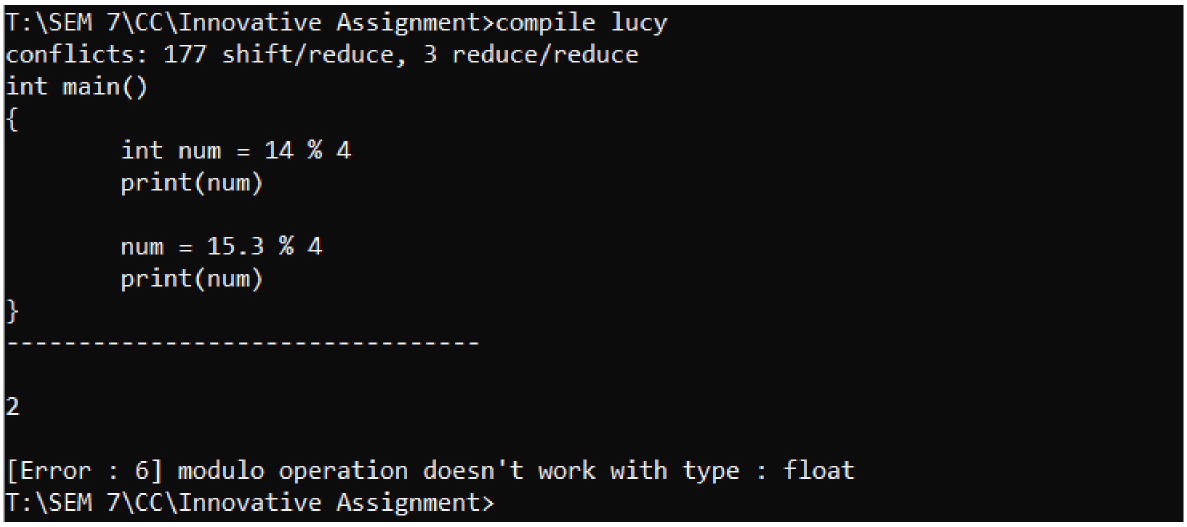
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        string num = "25"
        float num_2 = ~num

        // this is line no 6
}
---------------------------------


[Error : 4] bitwise not operation doesn't work with type : string
```



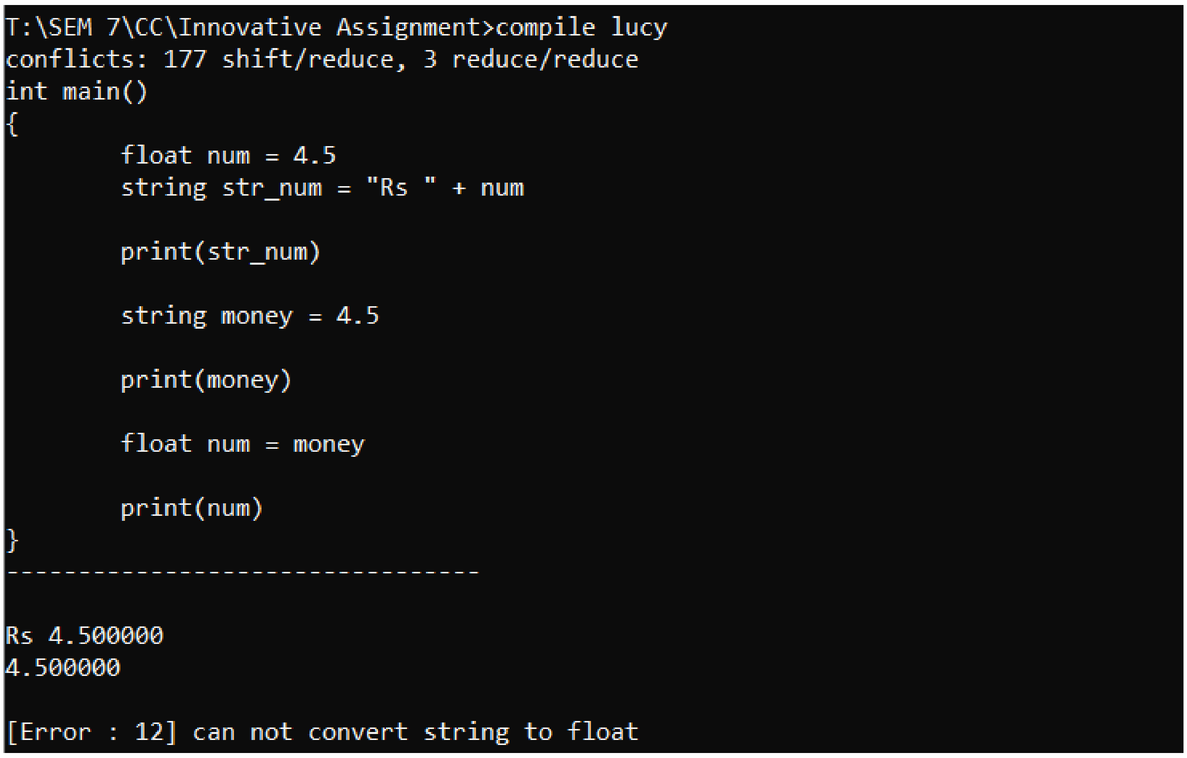
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        boolean flag1 = 4 > 5
        boolean flag2 = "ABC" < "ABD"
        boolean flag3 = "ABC" != 'ABC'

        print("flag1 : " + flag1)
        print("flag2 : " + flag2)
        print("flag3 : " + flag3)

        boolean flag4 = 45 < "46"
        print("flag4 : " + flag4)
}
---------------------------------

flag1 : false
flag2 : true
flag3 : false

[Error : 11] can not compare string with type : int
T:\SEM 7\CC\Innovative Assignment>
```



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        string full_name = "Narendra " + "Modi" + "(" + 2014 + ")"
        print(full_name)

        string first_name = "Narendra Modi" - "Modi"
        print(first_name)
}
---------------------------------

Narendra Modi(2014)

[Error : 6] subtraction operation doesn't work with type : string
T:\SEM 7\CC\Innovative Assignment>
```

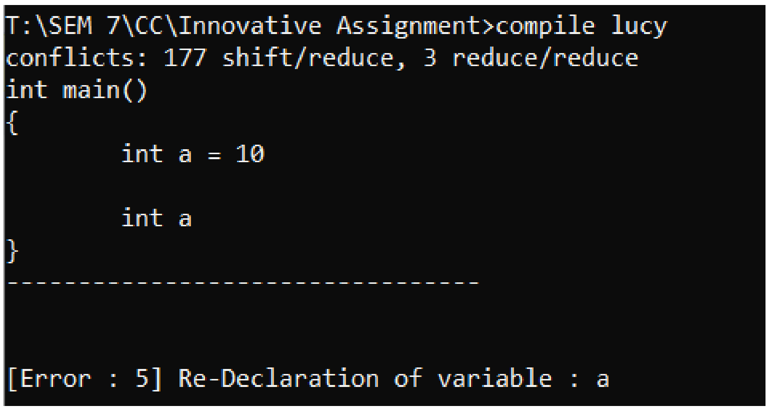


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int num = 14 % 4
        print(num)

        num = 15.3 % 4
        print(num)
}
---------------------------------

2

[Error : 6] modulo operation doesn't work with type : float
T:\SEM 7\CC\Innovative Assignment>
```

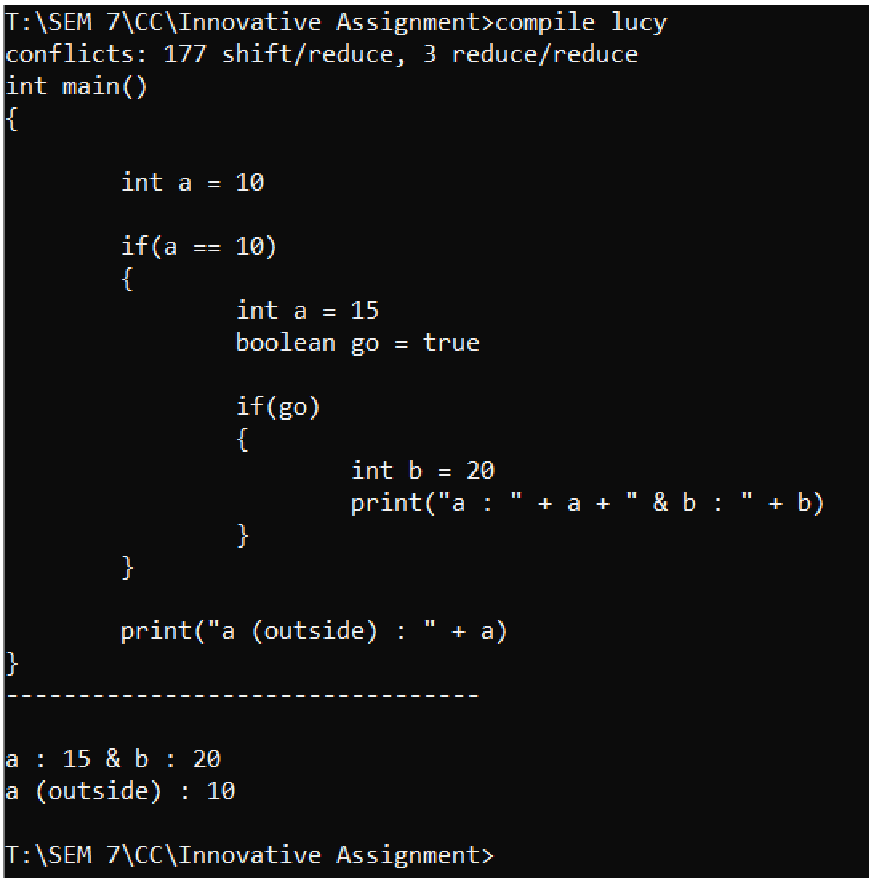


```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        float num = 4.5
        string str_num = "Rs " + num

        print(str_num)

        string money = 4.5

        print(money)

        float num = money

        print(num)
}
---------------------------------

Rs 4.500000
4.500000

[Error : 12] can not convert string to float
```

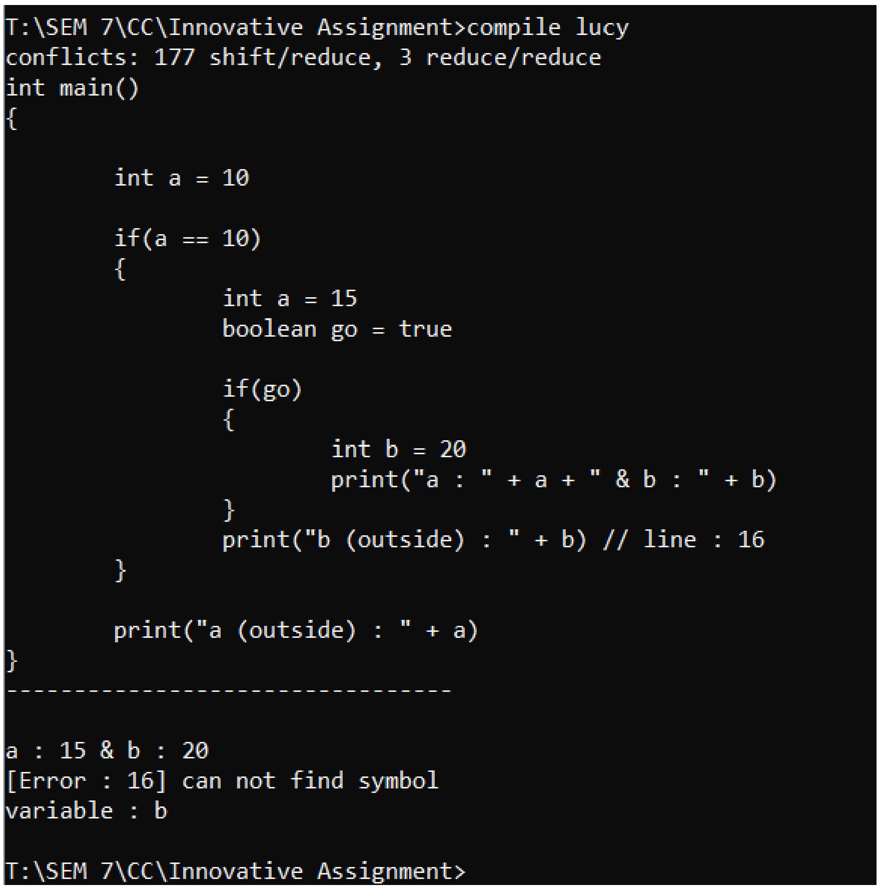
Scope Validity and Re-declaration Error



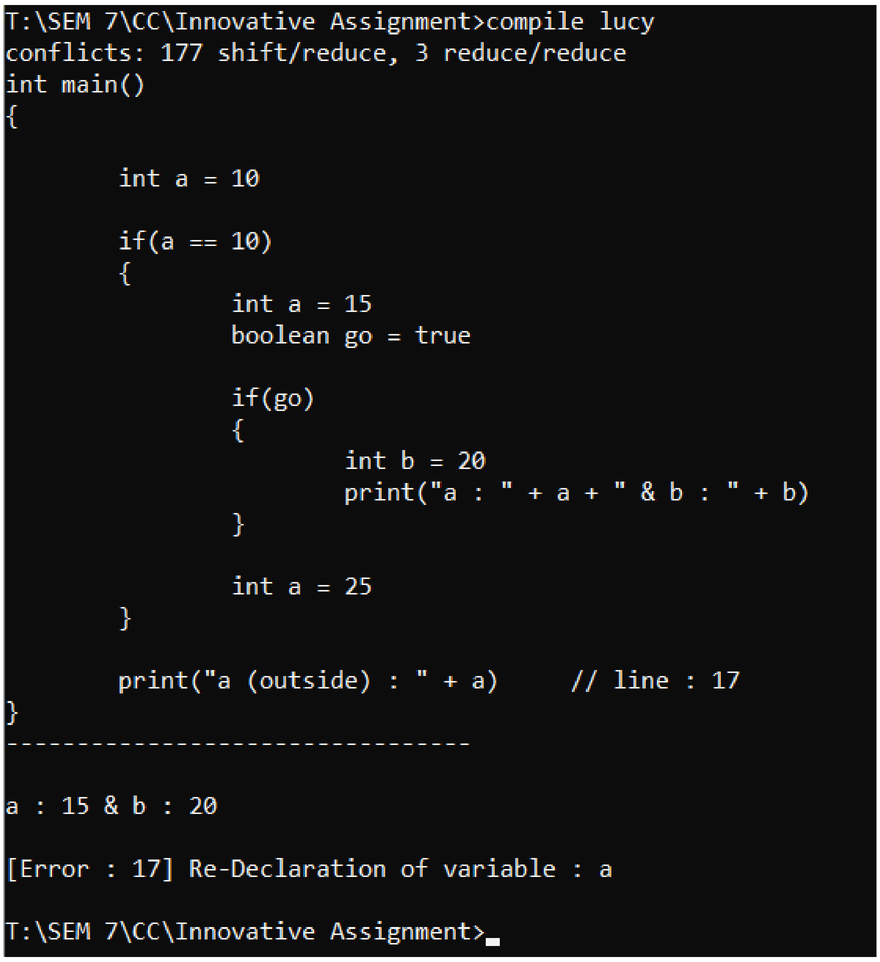
```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{
        int a = 10

        int a
}
---------------------------------


[Error : 5] Re-Declaration of variable : a
```



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{

        int a = 10

        if(a == 10)
        {
                int a = 15
                boolean go = true

                if(go)
                {
                        int b = 20
                        print("a : " + a + " & b : " + b)
                }
        }

        print("a (outside) : " + a)
}
---------------------------------

a : 15 & b : 20
a (outside) : 10

T:\SEM 7\CC\Innovative Assignment>
```



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{

        int a = 10

        if(a == 10)
        {
                int a = 15
                boolean go = true

                if(go)
                {
                        int b = 20
                        print("a : " + a + " & b : " + b)
                }
                print("b (outside) : " + b) // line : 16
        }

        print("a (outside) : " + a)
}
---------------------------------

a : 15 & b : 20
[Error : 16] can not find symbol
variable : b

T:\SEM 7\CC\Innovative Assignment>
```



```
T:\SEM 7\CC\Innovative Assignment>compile lucy
conflicts: 177 shift/reduce, 3 reduce/reduce
int main()
{

        int a = 10

        if(a == 10)
        {
                int a = 15
                boolean go = true

                if(go)
                {
                        int b = 20
                        print("a : " + a + " & b : " + b)
                }

                int a = 25
        }

        print("a (outside) : " + a)     // line : 17
}
---------------------------------

a : 15 & b : 20

[Error : 17] Re-Declaration of variable : a

T:\SEM 7\CC\Innovative Assignment>
```

Conclusion

In this assignment we have learnt how the compiler works on given program which is nothing but just a text. We are now able to understand different phases of other compilers and also have an idea to create new language. As in our project, the fine handling of even smaller rules of language are done by code written in C header files, that has cleared our basic concepts also. After doing the entire work it gives us great experience of how code written in different languages works just as an interface between machine and us. We appreciate the efforts of our faculties and programmers out there helping us through various platforms on internet.