FINAL ASSIGNMENT REPORT

<u>Directory Structure</u>:

Final project code is present in ***<u>'Assignment3'</u>*** folder. It contains files:

- ***compiler.l , compiler.y*** *:* Lex and yacc files.
- ***syntax_tree.h*** *:* This file contains all the node structures needed for building a syntax tree and also functions declarations for creating various nodes, AST printing, error checking functions , generating C code.
- ***syntax_tree.cpp:*** This file contains all the function definitions related to syntax tree creation , printing AST , semantic error checking , generation of C code.
- ***sym_table.h :*** This file contains structure of symbols stored in symbol table(map) and also symbol table function declarations like lookup , insertion and updation of symbol table. (update function is commented as it is not used due to no evaluation of syntax tree needed).
- ***sym_table.cpp:*** This file contains the symbol table definitions (lookup , inserting symbol, updating symbol)

<u>Instructions to run the code</u>:

Folder contains makefile. Using makefile, `` `make` `` *command will compile the code and produce an executable called 'compiler'.* Input file to the program should be given in the command itself as first argument while running i.e. `` `./compiler <input-file-path> <output-file-path>` `` (not like a redirection to executable).
Else using makefile, one can run the executable using command `` `make tests` ``. Then enter the input file name. (Input file(testcase) should be there in a folder named `` `testcases` ``). Output (C code) if exists, it will be present in outputs directory.

* There are 5 sample test cases in the folder `*testcases*`.

<u>Implementations in the code</u>:

All the implementations that were asked in the test are present.

1. Global declarations with 1D arrays are supported.
2. Support for all given relational and logical operators.
3. Support for conditional statements (if-then, if-then-else , do-while). Nested conditional statements are also supported.

4. Support for read and write statements. (Both write(expr) and write(“string_expr”) works. As per the grammar, ***write(expr) accepts only one expression value to print at a time***. And **s*tring expr should only contain alphabets and numbers but not any other characters like ‘=’ or ‘\n’ etc..*** And read can scan into integer and boolean data type variables. In case of boolean, if it is true: input should be given as 1 and 0 as false.
5. Error checking and printing with line numbers(syntax errors and semantic errors) , AST printing and generation of equivalent C code if there are no errors found(syntax , semantic errors).
6. All semantic errors were printed at once.
7. Support for functions, local declarations inside functions and function calls.

NOTE:

1. In case of syntax errors, <u>only one syntax error is being printed at once</u>. So syntax errors must be handled sequentially. <u>But incase of semantic errors, all are being printed at once with line numbers</u>.

2. Test cases uploaded in moodle have “\n” and “=” in write statements. But according to the grammar given initially, write statements accept only variables or strings containing only variables. So when executable is run with these test cases, it throws a syntax error as grammar doesn’t accept them.

3. Functions will be considered as valid only if they are declared in the global declaration block.(except main). Else function definitions of those which are not declared in the global declaration block will be thrown as semantic errors saying undeclared variable(function name) (except for main).

<u>Semantic errors that are handled:</u>

1. Undeclared variables , variables that were declared again were reported as errors.
2. In the case of arrays, index out of bounds were handled.
3. Array variables which are being accessed without index or integer/boolean variables being accessed with index as array variables were not allowed and reported as errors. Similarly for functions.

4. Division by zero error in case of ‘/’ or ‘%’ operations.
5. In case of functions, parameters type in function definition is compared to the type mentioned when declaring a function using symbol table. If not matched, reported it as an error.
6. Number of arguments in function declaration should be equal to its arguments in definition and also anytime when calling it(funcall).
7. Return type matching in both function declaration and definition.
8. Whenever integers and booleans are involved in the arithmetical or logical operations, a warning was thrown but the execution of the program won’t stop. (These are handled when we write explicitly ‘true+1’ but not the case with variable type).
9. In case of assigning boolean variables with integers or vice-versa , it is considered as normal (i.e. positive numbers=>true , zero/negative integers => False) like C language.

* In C language, arithmetic/logical operations between integers and booleans are allowed. Only a few cases of them were considered as warnings and most of the other cases were considered as normal in my code. (Not errors). Same thing is applicable even for function return statements. (return expression type).

<u>Testcases</u>: (`testcases` folder)

bin_search.txt: In bin_search.txt, there is a variable declared with the name ‘end’. According to the tokens given, ‘end’ is a token and not a variable. So it is considered as syntax error.

bubble.txt: There are no errors in the file. But this file accepts 5 inputs(5 integers). There is no statement like Enter the inputs in the given code. So when the cursor blinks, enter 5 integers. (It is not any infinite loop). Output will be ascending order of the input elements.(bubble sort).

Sum_Recursive,src.txt: No errors. Output will be 1275.

factorial_recursive.txt: In this testcase , ‘write’ statement accepts “\n” , “=” characters. But as per the grammar given to us, ‘write’ statement does not accept anything other than variables. Hence they are considered as syntax errors.

factorial_recursive_modified.txt: Syntax errors from previous file are removed. This file has no syntax and semantic errors. It accepts an input(integer) from the user to calculate its factorial.
Eg: Input 5 => Output: Factorial of 5 is 120.

-> If the input number is in a way that the factorial of it exceeds the INT_MAX, output will not be correct.As the equivalent C code generated contains only int,bool data types for any variables, output beyond the INT_MAX can’t be stored in a variable of INT type. (invalid output can be shown).

NOTE:

-> endwhile; , endif; , write(var_expr); , read(); , write(“string_expr”);   (string_expr contains only variables) are accepted by the grammar.