TEST2 REPORT

Directory Structure:

Exam code is present in *‘Test2’* 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 , evaluation functions.
- ***syntax_tree.cpp:*** This file contains all the function definitions related to syntax tree creation , printing AST , semantic error checking , evaluation of syntax tree.
- ***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.
- ***sym_table.cpp:*** This file contains the symbol table definitions (lookup , inserting symbol, updating symbol)

Instructions to run the code:

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>`* (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`)

* There are 3 sample test cases in the folder *`testcases`.*

Implementations in the code:

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

1. Global declarations with 1D arrays are supported.
2. Support for relational 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 string 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 , Evaluation of syntax tree and Symbol table values printing are part of the code.
6. All semantic errors were printed at once.

* Only the do-while conditional statement accepts ‘;’ at the end(i.e. endwhile;)

* Even Though in case of ‘read’ into a boolean variable accept only 1 as true and 0 as false but while assigning values to boolean variable, true and false can be used and if integers are assigned to boolean variables (positive values => true , zero / negative values => false).

* While printing the expression values using ‘write’ , if a boolean value is printed using write : it prints true as 1 and false as 0 but in the case of printing the symbol table values: boolean variable values are printed as true and false itself.

* Whenever input has to be given, cursor just blinks in the terminal for input. There is no statement like ‘enter input’.

Constructs which are not handled completely:

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

Semantic errors that are handled:

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.
4. Division by zero error in case of ‘/’ or ‘%’ operations.

5. 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).
6. 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.

<u>Semantic errors that are not handled:</u>
* 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)

<u>Testcases</u>: (`` `testcases` `` folder)
`test1.txt:` No semicolon after the endwhile => Syntax error near line 13.
As said before only syntax error can be shown at once. If multiple syntax errors need to be shown at once, we can add a line in the compiler.y file near 'stmt_list' production : error {yyerrok;} => this prints almost all errors. But it also prints errors in the line which are not correct. Hence I did not consider that case.

`test2.txt:` Semantic errors near lines 16,21.
Even though I printed the errors in those lines, I forgot to add the line number in 'cout' statement in syntax_tree.cpp for those two error cases. *In remaining error cases, line numbers were printed*. If line numbers need to be printed for these two cases, just add 'line_num' in the cout statement in lines 563,582 of syntax_tree.cpp file.

`test3.txt:` There are no errors in this file. Hence it should print all the values of the symbol table correctly. As it contains a read statement, when the cursor blinks in the terminal just enter an input integer for variable 'a'. ("Enter input" statement is not present).

* This test case contains integers, arrays , assign statements , read & write statements , conditional statements like if-then-else , do-while. When input was entered, symbol table values were being printed correctly.

`Atyam Lakshmi Nikhitha`
`112001009`