BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

Group No. 14

Compiler Construction (CS F363)
II Semester 2018-19
Compiler Project (Stage-2 Submission)
Coding Details
(April 14, 2019)

Instruction: Write the details precisely and neatly. Places where you do not have anything to mention, please write NA for Not Applicable.

1. IDs and Names of team members
    a. ID: 2016A7PS0005P Name: Tarun Kumar
    b. ID: 2016A7PS0081P Name: Vaibhav Maheshwari
    c. ID: 2016A7PS0087P Name: Varun Gupta
    d. ID: 2016A7PS0046P Name: Avichal Jain

2. Mention the names of the Submitted files (Include Stage-1 and Stage-2 both)

1. lexer.h	8. symbol.c	15. CodeGenerator.h	22. testcase4.txt
2. lexer.c	9. typeChecker.h	16. CodeGenerator.c	25. main1.txt
3. lexerDef.h	10. typeChecker.c	17. driver.c	26. main2.txt
4. parser.h	11. ast.h	18. grammar.txt	27. main3.txt
5. parser.c	12. ast.c	19. testcase1.txt	28. main4.txt
6. parserDef.h	13. icg.h	20. testcase2.txt	
7. symbol.h	14. Icg.c	21. testcase3.txt	

3. Total number of submitted files: 28 (All files should be in ONE folder named exactly as Group_#, # is your group number)
4. Have you compressed the folder as specified in the submission guidelines? (yes/no) Yes.

5. Status of Code development: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
    a. Lexer (Yes/No): Yes
    b. Parser (Yes/No): Yes
    c. Abstract Syntax tree (Yes/No): Yes
    d. Symbol Table (Yes/ No): Yes
    e. Type checking Module (Yes/No): Yes
    f. Semantic Analysis Module (Yes/ no): Yes (reached LEVEL 4 as per the details uploaded)
    g. Code Generator (Yes/No): Yes
6. Execution Status:
    a. Code generator produces code.asm (Yes/ No): Yes
    b. code.asm produces correct output using NASM for testcases (Main#.txt, #:1-4): Yes
    c. Semantic Analyzer produces semantic errors appropriately (Yes/No): Yes
    d. Type Checker reports type mismatch errors appropriately (Yes/ No): Yes
    e. Symbol Table is constructed (yes/no) Yes and printed appropriately (Yes /No): Yes
    f. AST is constructed (yes/ no) Yes and printed (yes/no) Yes
    g. Name the test cases out of 7 as uploaded on the course website for which you get the segmentation fault (testcase#.txt ; # 1-3 and Main@.txt ; @:1-4): NA

7. Data Structures (Describe in maximum 2 lines and avoid giving C definition of it)
    a. AST node structure: inh and syn pointers, information about the parent, children, line number and lexeme in case of terminals, nodeSymbol in case of non-terminals and parentnodeSymbol
    b. Symbol Table structure: A hash table of SymbolTableEntry which contains lexeme, token, and a set of attributes.
    c. Data structure for global variables: Inserted in global symbol table.
    d. Record type expression structure: Record ID inserted in global symbol table, with fields inserted in a hash table (of SymbolTableEntry) linked to the entry in global symbol table.
    e. Input parameters type structure: A hash table of SymbolTableEntry which contains lexeme, token, and a set of attributes.
    f. Output parameters type structure: A hash table of SymbolTableEntry which contains lexeme, token, and a set of attributes.
    g. Structure for maintaining the three address code(if created) : Quadruples
    h. Any other interesting data structure used: N.A.

8. Semantic Checks: Mention your scheme NEATLY for testing the following major checks
    a. Variable not Declared : Checked in all symbol tables to see if inserted previously.
    b. Multiple declarations: Checked in all symbol tables to see if inserted previously.
    c. Number and type of input and output parameters: Checked them in the input and output hash tables.
    d. assignment of value to the output parameter in a function: Checked for all the parameters in the output parameter list (by maintaining the output_par AST node) if they are on the LHS of assignment or function call statement or occur in an IO read statement.
    e. function call semantics: First checked if the no. of actual input and output parameters match the corresponding number of formal input and output parameters. Then if it matches, we match the type of all these actual parameters with the formal ones by invoking the input and output parameter hash tables of the function used in function call statement.
    f. type checking: Checked for all the operators and assignment statement if the type on the LHS equals the type on RHS (recursively in bottom-up manner).
    g. return semantics: Checked if the parameters being returned match exactly to the formal output parameter list (by maintaining the output_par AST node).
    h. Recursion: If a function has a function call of the same name as the current function.
    i. function overloading: Checked in global symbol table if function lexeme already inserted.
    j. 'while' loop semantics : Maintained a linked list of hash tables, with ith hash table having the variables used in the boolean expression of the while statement in the ith nested level of the while statements. Also, an boolean array is used whose ith index denotes if the while statement in the ith nested level has any one of its variables assigned in nested levels >= i.
    k. record data type semantics and type checking of record type variables: Maintained a different record number for each different type definition (i.e. not name equivalent). So, record-record semantics and type are dealt with that. When fields are used we invoke the record hash table, and deal with the semantic and type checking as described above.
    l. register allocation: N.A.
    m. Scope of variables and their visibility : Maintained a current symbol table and global symbol table and checked accordingly.
9. Compilation Details:
    a. Makefile works (yes/No): Yes
    b. Code Compiles (Yes/ No): Yes
    c. Mention the .c files that do not compile: N.A.
    d. Any specific function that does not compile: N.A.
    e. Ensured the compatibility of your code with the specified gcc version(yes/no) Yes

10. Driver Details: Does it take care of the options specified earlier?(yes/no): Yes

11. Specify the language features your compiler is not able to handle (in maximum one line): N.A.
12. Are you availing the lifeline (Yes/No): Yes
13. Write exact command you expect to be used for executing the code.asm using NASM simulator: (IMP: gcc-multilib package is required to build x86 programs on x64 system using gcc. Can be installed using 'sudo apt install gcc-multilib')
    nasm -f elf32 -o output.o code.asm
    gcc -m32 -o output output.o
    ./output

    Strength of your code(Tick the boxes where applicable): (a) correctness ✓ (b) completeness ✓ (c) robust ✓ (d) Well documented ✓ (e) readable ✓ (f) strong data structure ✓ (f) Good programming style (indentation, avoidance of goto stmts etc) ✓ (g) modular ✓ (h) space and time efficient ✓

14. Any other point you wish to mention: N.A.
15. Declaration: We, Vaibhav Maheshwari, Varun Gupta, Avichal Jain, Tarun Kumar declare that we have put our genuine efforts in creating the compiler project code and have submitted the code developed by us. We have not copied any piece of code from any source. If our code is found plagiarized in any form or degree, we understand that a disciplinary action as per the institute rules will be taken against us and we will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

    Date: 15 April, 2019