**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

Batch No. :

**Group No.11**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**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

ID: 2016A7PS0095P Name: Aditi Agarwal

ID: 2016A7PS0057P Name: Nikki Gupta

ID: 2016A7PS0038P Name: Aditya Laddha

ID: 2016A7PS0097P Name: Sahil Ranadive

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

1 lexer.c 7 ast.c 13 typechecker.h 19 NTerminals.txt

2 parser.c 8 symboltable.c 14 codegen.c 20 Grammar.txt

3 lexer.h 9 symboltable.h 15 semanticanalyzer.c 21 tokenID.txt

4 parser.h 10 symboltableDef.h 16 semanticanalyzer.h 22 lexerDFA.pdf

5 lexerDef.h 11 ast.h 17 codegen.h 23 supportgrammar.pdf

6 parserDef.h 12 typechecker.c 18 driver.c 24 languagespecifications.pdf 25 First.txt 26Follow.txt

27 makefile

1. Total number of submitted files: \_\_27\_\_ (All files should be in ONE folder named exactly as Group\_#, # is your group number)
2. Have you compressed the folder as specified in the submission guidelines? (yes/no) \_\_\_yes\_\_\_
3. Status of Code development: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
   1. Lexer (Yes/No): \_\_\_YES\_\_\_
   2. Parser (Yes/No):\_\_\_YES\_\_\_
   3. Abstract Syntax tree (Yes/No):\_\_\_YES\_\_\_
   4. Symbol Table (Yes/ No): \_\_\_YES\_\_\_
   5. Type checking Module (Yes/No): \_\_\_YES\_\_\_
   6. Semantic Analysis Module (Yes/ no): \_\_\_YES\_\_\_ (reached LEVEL \_\_\_3\_ as per the details uploaded)
   7. Code Generator (Yes/No): \_\_\_YES\_\_\_
4. Execution Status:
   1. Code generator produces code.asm (Yes/ No): \_\_\_YES\_\_\_
   2. code.asm produces correct output using NASM for testcases (Main#.txt, #:1-4): 1,2,3,4
   3. Semantic Analyzer produces semantic errors appropriately (Yes/No):\_\_\_\_\_YES\_\_\_\_\_\_\_
   4. Type Checker reports type mismatch errors appropriately (Yes/ No):\_\_\_YES\_\_\_
   5. Symbol Table is constructed (yes/no):\_\_\_YES\_\_\_and printed appropriately (Yes /No):\_\_\_YES\_\_\_
   6. AST is constructed (yes/ no) \_\_\_YES\_\_\_and printed (yes/no) \_\_\_YES\_\_\_
   7. 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): NONE
5. Data Structures (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure: The AST node contains a the lexeme of the construct/object, its line number in the original code and an integer id which contains information about the token. It also contains pointers to its parent, right sibling and first child.
   2. Symbol Table structure: Symbol table is implemented as an array of linked lists where each array element describes a function and each element of the linked list in the array element is a variable declared in its scope.
   3. Data structure for global variables: It contains an integer to show the type of the variable, the lexeme for the variable(instance), its width, offset, a string which defines that the scope of this variable is global, the line number of the variable in the original code , and a string which is initialed to the type of record(TK\_RECORDID) if variable is of type record.
   4. Record type expression structure: It contains a string to store the name of the record definition, an integer width to store the width of the entire record, a pointer to a list containing the fields of the record and the line number where the record is defined.
   5. Input parameters type structure: uses same structure as global variables
   6. Output parameters type structure: uses same structure as global variables
   7. Structure for maintaining the three address code(if created) : Not Created
   8. Any other interesting data structure used : 1. each symbol table element contains separate lists for input parameters, output parameters and local variables along with the scope of the function.
6. Semantic Checks: Mention your scheme NEATLY for testing the following major checks
   1. Variable not Declared : Check for variable(when we are not declaring) in the input parameter list, output parameter list and local variables of the function in which it is called along with the list of global variables. If it does not appear in any list then variable is not declared.
   2. Multiple declarations: If we are checking for declarations and variable appears in input parameter list, output parameter list or local variables of function as already declared or in global variable list. Give error.
   3. Number and type of input and output parameters: Traverse through function call statements to match type and count actual and formal parameters(stored in symbol table),both input and output.
   4. assignment of value to the output parameter in a function: not checked
   5. function call semantics: checking for number and type of actual and formal parameters in the function call. Function being called should be declared previously.
   6. type checking: left hand side and right hand side of assignment statements checked. Type checking in boolean expressions(both conditional and iterative statements) done.
   7. return semantics: not done
   8. Recursion : check if function call statement is defined inside the function itself. This is not allowed and throws an error.
   9. function overloading: if the function is already defined in the symbol table, redefining the function throws an error
   10. 'while' loop semantics : not implemented
   11. record data type semantics and type checking of record type variables : checking for type of all the fields should match with the types defined in record definition. Record should be defined previously(before its use). Type checking for assignment to record or record fields done.
   12. register allocation: 14 temporary registers have been used each storing a double
   13. Scope of variables and their visibility : for all statements in a function checking if variable used has been declared or not, using scope from symboltable
7. Compilation Details:
   1. Makefile works (yes/No):\_\_\_YES\_\_\_
   2. Code Compiles (Yes/ No):\_\_\_YES\_\_\_
   3. Mention the .c files that do not compile: NONE
   4. Any specific function that does not compile: NONE
   5. Ensured the compatibility of your code with the specified gcc version(yes/no):\_\_\_YES\_\_\_
8. Driver Details: Does it take care of the options specified earlier?(yes/no):\_\_\_YES\_\_\_
9. Specify the language features your compiler is not able to handle (in maximum one line): scalar multiplication of records not handled
10. Are you availing the lifeline (Yes/No): \_\_\_YES\_\_\_
11. Write exact command you expect to be used for executing the code.asm using NASM simulator – $nasm -f elf64 code.asm

$gcc code.o -o code

$./code

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

(a) (c) (d) (e) (f) (g) (h) (i) are applicable

1. Any other point you wish to mention: The project although was quite time consuming but the learning we received was worth the time spent.
2. Declaration: We, Aditi Agarwal, Nikki Gupta, Aditya Laddha and Sahil Ranadive 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: 15/04/2019

(Not to exceed beyond 3 pages)