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

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**II Semester 2017-18**

**Compiler Project (Stage-2 Submission)**

**Coding Details**

**(April 20, 2018)**

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

1. ID Number: 2015A7PS0083P

Name: Deepak Guneja

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

1: Grammar.txt 2: abstractSyntaxTree.c 3: abstractSyntaxTree.h

4: abstractSyntaxTreeDef.h 5: allKeywords.txt 6: codeGenerator.c

7: codeGenerator.h 8: driver.c 9: hashTableLexer.c 10:hashTableLexer.h

11: header.h 12: intermediateCodeGen.c 13: intermediateCodeGen.h

14: intermediateCodeGenDef.h 15: lexer.c 16: lexer.h

17: lexerDef.h 18: makefile 19: parser.c 20: parser.h

21: parserDef.h 22: readPrint.asm 23: symbolTableAndTypeChecker.c

24: symbolTableAndTypeChecker.h 25: symbolTableAndTypeCheckerDef.h

26: codingDetails.docx 27-30 : c1.txt - c4.txt 31-50: testcase1.txt – testcase20.txt

1. Total number of submitted files: 50 (All files should be in ONE folder named exactly as your ID)
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 4 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 (C#.txt, #:1-3): Yes
   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 9 as uploaded on the course website for which you get the segmentation fault (testcase#.txt ; # 1-6 and c@.txt ; @:1-3): None
5. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure : parent, children, next, varType t, token\* tkn, ASTnodeSym sym, symTablePtr, code, addr, trueLabel, falseLabel, nextLabel
   2. Symbol Table structure: varType t, char\* lexeme, char\* funcScopeName, next pointer, idPtr (if entry is id then pointer to this struct idPtr) else funcPtr (if entry is funID)
   3. Matrix type expression structure: parent: MATRIXDECL (a dummy node)

Its children are ROW → ROW →ROW … (where ROW is a dummy node)

and each ROW has children NUM1→ NUM2→ NUM3...

* 1. Input parameters type structure: linked list of children having parent as IN\_PARAM (a dummy node) and each parameter is child, where each child is (type, id)
  2. Output parameters type structure: linked list of children having parent as OUT\_PARAM (a dummy node) and each parameter is child, where each child is (type, id)
  3. Structure for maintaining the three address code(if created) : Created 4 address code, a quadruple which has structure (result, op, arg1, arg2), where result, arg1 and arg2 are ASTnode and op is the operator to be done.

1. **Semantic Checks:** Mention your scheme NEATLY for testing the following major checks (in not more than 5-10 words)[ Hint: You can use simple phrases such as 'symbol table entry empty', 'symbol table entry already found populated', 'traversal of linked list of parameters and respective types' etc.]
   1. Variable not Declared : symbol Table entry empty
   2. Multiple declarations: symbol table entry already found populated
   3. Number and type of input and output parameters: traversal of list in the symbol table entry of that function, and verifying input and output parameters
   4. assignment of value to the output parameter in a function: used a flag, which is set to 0 if output parameter not initialised
   5. function call semantics: checking for that funID in symbolTable and then if exists, traversal the list in the symbol table entry of that function, and verifying input and output parameters
   6. type checking : traversing AST only once and populating symbol table and checking type by bottom up
   7. return semantics: checking the number and type of output parameters of the function called, by looking at its symbol table entry
   8. Recursion : checking the symbol table stack and finding if the function is already loaded
   9. module overloading: symbol table entry already found populated
   10. 'If' semantics : type checking the boolean expression by bottom up traversing AST
   11. Matrix semantics and type checking of matrix type variables: calculating size of a row by traversing AST, and if it does not match previous row size, giving error otherwise update size in symbolTable.

And if 2 matrices then checking their size in their symbol table and if equal then doing the given operation

* 1. register allocation (your manually selected heuristic) : AX, BX for general purpose, and EBP for stack
  2. Scope of variables and their visibility : by looking at stack of symbol table and checking if they exist in which symbol table

1. **Compilation Details**:
   1. Makefile works (yes/No): yes
   2. Code Compiles (Yes/ No): yes
   3. Mention the .c files that do not compile: NA
   4. Any specific function that does not compile: NA
   5. Ensured the compatibility of your code with the specified gcc version(yes/no): yes
2. **Driver Details**: Does it take care of the options specified earlier?(yes/no): yes
3. Specify the language features your compiler is not able to handle (in maximum one line): NA
4. Are you availing the lifeline (Yes/No): Yes
5. Write exact command you expect to be used for executing the code.asm using NASM simulator [We will use these directly while evaluating your NASM created code]

: If the output file is code.asm →

nasm -f elf code.asm [Press Enter]

ld -m elf\_i386 code.o -o code [Press Enter]

./code [Press Enter]

1. **Strength of your code**(Strike off where not applicable): (a) correctness (b) completeness (c) robustness (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
2. Any other point you wish to mention: NA
3. **Declaration:** I, Deepak Guneja, declare that I have put my genuine efforts in creating the compiler project code and have submitted the code developed by me. I have not copied any piece of code from any source. If my code is found plagiarized in any form or degree, I understand that a disciplinary action as per the institute rules will be taken against me and I will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

Sign:Deepak Guneja

ID: 2015A7PS0083P

Name: Deepak Guneja

Date: 21/4/18

-------------------------------------------------------------------------------------------------------------------------------------------------

/\*not to exceed three pages\*/