**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: \_\_\_\_\_\_2014A3A70302P\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name: \_\_\_\_\_\_T Dinesh Ram Kumar\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

1\_abstractDef.h \_\_\_\_\_ 7\_\_intermediatecode.c\_\_\_ 13\_\_\_parser.c\_\_\_\_\_\_\_ 19\_\_\_stable.h\_\_\_\_

2 abstractnstableDef.h 8\_\_intermediatecode.h\_\_\_ 14\_\_\_parserDef.h\_\_\_\_\_ 20\_\_strfunctions.asm\_\_

3\_\_\_ast.c \_\_\_\_\_\_\_\_\_\_\_\_\_ 9\_\_\_\_\_lexer.c\_\_\_\_\_\_ 15\_\_\_\_parser.h\_\_\_\_ 21\_\_\_symbolhashs.c\_\_\_

4\_\_commentfree.c\_\_\_\_\_ 10\_\_\_\_lexerDef.h\_\_\_\_\_\_\_\_ 16\_\_\_\_parserStack.c\_\_\_\_ 22\_\_symbolhashs.h\_\_\_\_\_

5\_\_driver.c\_\_\_\_\_\_\_\_\_\_\_\_ 11\_\_\_\_\_lexer.h\_\_\_\_\_\_\_ 17\_\_\_parserStack.h\_\_\_\_\_ 23\_\_\_ symboltable.c\_\_\_\_\_

6\_\_grammar-final.txt\_\_\_\_ 12\_\_\_ Makefile\_\_\_\_\_\_\_ 18\_\_semanticanalysis.c\_ 24\_\_\_codingDetails.docx\_\_\_

1. Total number of submitted files: 24 + 24 test-cases (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):\_\_\_\_\_ALL\_CORRECT\_\_\_\_
   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):\_\_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure: Two types of nodes used. Each pointing to array of children (if any). It is union of various structures (each for corresponding non-terminal). Entries for facilitating semantic checks, IR and code generation.
   2. Symbol Table structure: Contains two Hash Tables (using Separate Chaining) for functions and variables. Points to static parent for static scoping.
   3. Matrix type expression structure: Single AST Node pointing to 2D array of elements.
   4. Input parameters type structure:\_Array of Input types maintained by function definition AST Node.
   5. Output parameters type structure:\_Array of Output types maintained by function definition AST node.
   6. Structure for maintaining the three address code(if created) :\_\_Integrated as a part of AST node\_(No additional structure used)\_\_
6. **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.]

NOTE: ALL OF THEM ACHIEVED BY USING SYMBOL TABLE RECORD ENTRIES and AST.

* 1. Variable not Declared : corresponding symbol table entry empty in current and ancestor scopes.
  2. Multiple declarations: symbol table entry already found populated
  3. Number and type of input and output parameter: Parameters must have unique names.
  4. assignment of value to the output parameter in a function : Types and sizes must match.
  5. function call semantics: Initialized input arguments. Types, sizes and count must match.
  6. type checking : Strict type checking (Everywhere). Operands same type. Operand support for that type.
  7. return semantics: output parameters must be initialized
  8. Recursion : Function must not involve call to itself or any parent and also must only functions declared before it.
  9. module overloading: Symbol table entry exists already in visible scope.
  10. 'If' semantics : Operands must be of same and valid type in Boolean expression
  11. Matrix semantics and type checking of matrix type variables: Column size must match across rows
  12. register allocation (your manually selected heuristic) : Load, Operation and Store technique used.
  13. Scope of variables and their visibility: Traversal of Symbol Table Tree Towards Root 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\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Are you availing the lifeline (Yes/No): \_\_YES\_\_\_
2. 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]

\_\_\_\_\_\_\_nasm -felf64 -o out.o code.asm && gcc out.o -no-pie -o out && ./out\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Strength of your code**(Strike off where not applicable): (a) correctness (b) completeness (c) robustness (d) Moderately documented (e) moderately readable (f) very strong data structure (f) Good programming style (indentation, avoidance of goto stmts etc) (g) modular (h) highly space and time efficient
2. Any other point you wish to mention: \_\_Aggressive initialization checks used. Function definition must involve operation that fixes matrix size. Matrix is single node. AST node points of array of AST children nodes. Output well presented (indented and colored). AST and Parse Tree use different nodes.
3. **Declaration:** I, \_T DINESH RAM KUMAR\_ (your name) 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:\_\_DINESH\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID\_\_\_2014A3A70302P\_\_\_\_\_\_\_\_\_

Name:\_\_T DINESH RAM KUMAR\_\_\_\_\_

Date: \_21/4/2018\_

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

/\*not to exceed three pages\*/