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

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

Group Number

50

**II Semester 2019-20**

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

**Coding Details**

**(April 20, 2020)**

*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: 2017A7PS0145P Name: Aniruddha Mahajan

ID: 2017A7PS0146P Name: Ravindra Singh Shekhawat

ID: 2017A7PS0162P Name: Shreyas Srikrishna

ID: 2017A7PS0182P Name: Chetan Sharma

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

1\_\_\_\_ast.c\_\_\_\_\_\_\_\_\_\_ 7\_\_\_\_functionCheck.c\_\_\_ 13\_\_offSetComputer.c\_ 19\_\_\_semantic.h\_\_\_\_

2\_\_\_\_\_ast.h\_\_\_\_\_\_\_\_\_\_\_ 8\_\_\_functionCheck.h\_\_\_\_ 14\_\_\_offsetComputer.h 20 semanticDef.h

3\_\_\_\_\_astDef.h\_\_\_\_\_\_\_\_\_ 9\_\_\_\_\_\_Grammar.txt\_\_\_\_\_ 15\_\_\_\_\_parser.c 21\_\_\_\_\_symboltable.c\_

4\_\_\_\_codeGeneration.c\_\_ 10\_\_\_\_\_lexer.c\_\_\_\_\_\_\_\_\_\_ 16\_\_\_\_\_\_parserDef.h 22\_\_\_\_\_\_symboltable.h

5\_\_\_ codeGeneration .h\_\_\_\_ 11\_\_\_lexerDef.h\_\_\_\_\_\_\_ 17\_\_\_\_\_parser.h 23\_\_\_\_\_symboltableDef.h

6\_\_\_\_\_driver.c\_\_\_\_\_\_\_\_\_\_\_ 12\_\_\_\_\_lexer.h\_\_\_\_\_\_\_\_\_ 18\_\_\_\_\_semantic.c 24\_\_\_\_\_typeExtractor.c

25 typeExtractorDef.h 26 typeExtractor.h 27. t1.txt-t10.txt 28. c1-c11.txt 29. makefile 29. Coding Details ProForma

1. Total number of submitted files: \_\_\_\_48\_\_\_\_\_\_\_ (All files should be in **ONE** folder named exactly as Group number)
2. Have you mentioned names and IDs of all team members at the top of each file (and commented well)? (Yes/ no) \_\_\_\_\_Yes\_\_\_\_\_ [Note: Files without names will not be evaluated]
3. Have you compressed the folder as specified in the submission guidelines? (yes/no)\_\_\_\_\_Yes\_\_\_\_\_\_\_\_\_\_
4. **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\_\_\_\_\_\_\_\_\_\_
5. **Execution Status**:
   1. Code generator produces code.asm (Yes/ No):\_\_\_\_\_\_\_\_\_\_\_Yes\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   2. code.asm produces correct output using NASM for testcases (C#.txt, #:1-11): Correct for all except c3.txt,c10.txt,c11.txt
   3. Semantic Analyzer produces semantic errors appropriately (Yes/No):\_\_\_\_\_Yes\_\_\_\_\_\_\_
   4. Static Type Checker reports type mismatch errors appropriately (Yes/ No):\_\_\_\_\_\_Yes\_\_\_\_\_\_\_\_
   5. Dynamic type checking works for arrays and reports errors on executing code.asm (yes/no):\_\_\_\_\_\_\_\_\_\_Yes\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   6. Symbol Table is constructed (yes/no)\_\_Yes\_\_and printed appropriately (Yes /No):\_\_\_\_Yes\_\_\_\_\_\_\_\_\_\_\_\_
   7. AST is constructed (yes/ no) \_\_\_\_\_Yes\_\_\_\_\_\_\_and printed (yes/no) \_\_\_Yes\_\_\_\_\_\_\_
   8. Name the test cases out of 21 as uploaded on the course website for which you get the segmentation fault (t#.txt ; # 1-10 and c@.txt ; @:1-11):\_\_\_None\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure: AST node contains 1) a union of different types of possible AST nodes. 2) a tag field to identify them. 3) a pointer to first child of this node. 4) a pointer to right sibling of this node. 5) a pointer to parent node of this node. 6) a next pointer (used in making linked list of such nodes).
   2. Symbol Table structure: Symbol table is implemented in form of tree of symbol tables. For each node we have we have 1) A hash table in which all symbol table entries corresponding to current scopes are hashed. 2) A pointer to first child of the current table (inward scoping). 3) A parent pointer to store parent symbol table of current table. 4) A right sibling pointer to store right sibling of the current table. 5) A union to store scope of this table.
   3. array type expression structure: Array type expression is stored in a structure having 2 integer fields to store high and low indices (if static) and 2-character array fields to store high and low indices (if dynamic). It also contains an Enum storing primitive type of array.
   4. Input parameters type structure: Input parameters are treated same as a variable except the have additional Boolean fields which is used to check whether they are input parameters or not and are they overshadowed.
   5. Output parameters type structure: output parameters are treated same as a variable except the have an additional Boolean field which is used to check whether they are output parameters or not.
   6. Structure for maintaining the three-address code (if created): N.A.
7. **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: When lookup in symbol table variable not found.
   2. Multiple declarations: Checked during symbol table creation if variable is found in lookup.
   3. Number and type of input and output parameters: looking up for each input and output parameter in symbol table.
   4. assignment of value to the output parameter in a function: By making a Boolean field in function entry of symbol table.
   5. function call semantics: by checking types and numbers of input and output parameters of function from symbol table.
   6. static type checking: By checking high and low values associated with the array in symbol table entry of that array.
   7. return semantics: Having a Boolean parameter for each of output parameters which is set true when-ever the parameter is assigned a value. And then traversing list of output parameters to check if any field is left false.
   8. Recursion: By traversing parent pointers in AST till any module is reached. If this reached module is same as that of module for which we are checking then recursion.
   9. module overloading: During Symbol Table creation if symbol table entry corresponding to that module is found then module overloading is there.
   10. 'switch' semantics: By traversing each case nodes and checking that according to rules. Checks are also done for type of switching variable.
   11. 'for' and 'while' loop semantics: By traversing each statement of for loop we checked the assignment of for loop iterating variable. In case of while loop we used a flag variable which is set to true if any variable in while loop condition is in LHS of any assignment statement. Checks related to for loop variable are also implemented.
   12. handling offsets for nested scopes: A global variable is made which is set to 0 at the start of every module and then by traversing all declare statements this offset variable is incremented by width of variable here assigning of offset is also done.
   13. handling offsets for formal parameters: So global offset variable is set to 0 before traversing list of formal parameters, for each parameter offset is assigned as current value of global offset variable and then global variable is incremented accordingly.
   14. handling shadowing due to a local variable declaration over input parameters: A bool variable is present in each input parameter entry as well as a next pointer. Whenever a overshadowing happens the symbol table entry corresponding to new variable is assigned to next pointer
   15. array semantics and type checking of array type variables: From symbol table entry of each array variable the type related information is extracted then compared according to rules described in Nalanda notice.
   16. Scope of variables and their visibility: Each Id Entry in symbol table has pointer to it’s table. Scope of each table is stored in symbol table entry corresponding to that table.
   17. computation of nesting depth: Traversed the AST and incremented the nesting level by 1 as entered into any of these constructs - while, for and switch.
8. Code Generation:
   1. NASM version as specified earlier used (Yes/no):\_\_\_\_\_\_Yes\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   2. Used 32-bit or 64-bit representation:\_\_\_\_\_\_\_\_\_\_\_\_\_64-bit\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. For your implementation: 1 memory word = \_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_(in bytes)
   4. Mention the names of major registers used by your code generator:

* For base address of an activation record: \_\_rbp\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* for stack pointer:\_\_\_\_\_rsp\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* others (specify):\_\_\_\_\_rax, rcx, rdi, rsi, rbx\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): \_\_\_\_\_\_\_2 locations\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_(in bytes)

size(real): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4 locations\_\_\_ (in words/ locations), \_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_(in bytes)

size(booelan): \_\_\_\_\_\_\_\_\_\_\_\_1 location\_\_\_(in words/ locations), \_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_(in bytes)

* 1. How did you implement functions calls?(write 3-5 lines describing your model of implementation) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Specify the following:
     + Caller's responsibilities:\_\_\_\_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     + Callee's responsibilities:\_\_\_\_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. How did you maintain return addresses? (write 3-5 lines): \_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? \_\_\_\_\_\_\_\_\_\_\_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. How is a dynamic array parameter receiving its ranges from the caller? \_\_\_NA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  3. What have you included in the activation record size computation? (local variables, parameters, both): \_\_\_only local variables\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  4. register allocation (your manually selected heuristic) :\_ax(for integer expr) and al (for boolean expr) holds the result of expressions on rhs of assignment stmt / expressions of while loop condition\_\_\_\_\_\_\_
  5. Which primitive data types have you handled in your code generation module?(Integer, real and Boolean ): Integer and Boolean
  6. Where are you placing the temporaries in the activation record of a function? In the end, after all local variables have been placed\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Compilation Details**:
   1. Makefile works (yes/No):\_\_\_\_Yes\_\_\_\_\_\_\_
   2. Code Compiles (Yes/ No):\_\_\_\_\_Yes\_\_\_\_\_\_\_\_\_
   3. Mention the .c files that do not compile:\_\_\_\_\_\_\_\_\_N.A.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. Any specific function that does not compile:\_\_\_\_\_\_\_\_\_\_\_\_N.A.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   5. Ensured the compatibility of your code with the specified versions [GCC, UBUNTU, NASM] (yes/no)\_\_\_\_\_\_Yes\_\_\_\_\_\_
2. Execution time for compiling the test cases [lexical, syntax and semantic analyses including symbol table creation, type checking and code generation]:
   * 1. t1.txt (in ticks) \_\_\_\_\_\_\_\_2296\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_0.002296\_\_\_
     2. t2.txt (in ticks) \_\_\_\_\_\_\_\_2576\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_0.002576\_\_\_\_\_
     3. t3.txt (in ticks) \_\_\_\_\_\_\_\_4051\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_0.004051\_\_\_\_\_\_
     4. t4.txt (in ticks) \_\_\_\_\_\_\_\_\_3719\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_0.003719\_\_\_\_\_\_\_
     5. t5.txt (in ticks) \_\_\_\_\_\_\_\_\_3716\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_0.003716\_\_\_\_\_\_\_\_
     6. t6.txt (in ticks) \_\_\_\_\_\_\_\_\_3907\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_0.003907\_\_\_\_\_
     7. t7.txt (in ticks) \_\_\_\_\_\_\_\_\_\_4052\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_0.004052\_\_\_\_\_\_\_\_\_\_
     8. t8.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_2870\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_0.002870\_\_\_\_\_\_\_\_\_\_\_
     9. t9.txt (in ticks) \_\_\_\_\_\_\_\_\_\_7563\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_0.007563\_\_
     10. t10.txt (in ticks) \_\_\_\_\_\_\_\_\_2748\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_ 0.002748\_\_\_\_
3. **Driver Details**: Does it take care of the **TEN** options specified earlier?(yes/no):\_\_\_\_\_Yes\_\_\_\_\_\_
4. Specify the language features your compiler is not able to handle (in maximum one line)

Code Generation for multiple modules.

1. Are you availing the lifeline (Yes/No): \_\_\_No\_\_\_\_\_\_\_\_\_\_\_
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 -f elf64 code.asm

gcc -no-pie code.o -o code

./code

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: \_\_\_\_N.A.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Declaration: We, Aniruddha Mahajan, Ravindra Singh Shekhawat, Shreyas Srikrishna, Chetan Sharma declare that we have put our genuine efforts in creating the compiler project code and have submitted the code developed only by our group. 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. [Write your ID and names below]

ID: 2017A7PS0145P Name: Aniruddha Mahajan

ID: 2017A7PS0146P Name: Ravindra Singh Shekhawat

ID: 2017A7PS0162P Name: Shreyas Srikrishna

ID: 2017A7PS0182P Name: Chetan Sharma

Date: \_\_20/04/2020\_\_\_\_\_\_\_\_\_\_\_\_

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

Should not exceed 6 pages.