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

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**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: **2017A7PS0031P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Anirudh Goyal**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0069P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Anish SS Kumar**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0122P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name:  **Rohit Jain**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0166P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Aditya Saxena**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

**1 arrayOfLinkedList.c 7 c3.txt 13 codegeneration.c 19 lexer.c 25 parser.c**

**2 ast.c 8 c4.tx 14 driver.c 20 lexer.h 26 parser.h**

**3 ast.h 9 c5.txt 15 FunctionTable.c 21 lexerDef.h 27 parserDef.h**

**4 astDef.h 10 c6.txt 16 FunctionTable.h 22 makefile 28 stack.c**

**5 c1.txt 11 c8.txt 17 grammar.txt 23 mappingTable.c 29 stpopulate.c**

**6 c2.txt 12 c9.txt 18 Hash\_table.c 24 mappingTable.h 30 SymbolTable.c**

**31 SymbolTable.h 32 t1.txt 33 t2.txt 34 t3.txt 35 t4.txt**

**36 t5.txt 37 t6.txt 38 t7.txt 39 t8.txt 40 t9.txt**

**41 t10.txt 42 typeextraction.c 43 coding\_Details(stage 2).docx**

1. Total number of submitted files: \_\_\_\_\_**43**\_\_\_\_\_ (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): \_\_\_\_\_**1-9**\_\_\_\_\_\_\_\_\_\_\_
   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):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**c11.txt**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure

**contains trivial information like terminal/ no-terminal name, lexeme, line\_no, value if number, child pointers, no\_child, pointer to corresponding symbol table and scope information**

* 1. Symbol Table structure:

**Implemeted as a hash table vo key value pair where key is the variable name and Value is a structure (STValue) that contains type, offset, scope and other relevant information. Nesting Scope is implemented via chaining of symbol tables by using child and parent pointers**

* 1. array type expression structure:

**contains 4 parameters: dtype (primitive data type that is integer, real or boolean), begin and end (to store the array beginning and end of static array) and isStatic (=1 if array is static and 0 if it is a dynamic array)**

* 1. Input parameters type structure:

**contains 5 parameters: dtype (primitive data type that is integer, real or boolean), isArray (=1 if an array otherwise zero), begin and end (to store the array beginning and end in case of static array) and isStatic (=1 if array is static and 0 if it is a dynamic array, =1 if not an array)**

* 1. Output parameters type structure:

**contains 5 parameters: dtype (primitive data type that is integer, real or boolean), isArray (=1 if an array otherwise zero), begin and end (to store the array beginning and end in case of static array) and isStatic (=1 if array is static and 0 if it is a dynamic array, =1 if not an array)**

* 1. Structure for maintaining the three address code(if created) :\_\_\_\_**Not created**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 : **undefined reference to variable x**
   2. Multiple declarations: **multiple declarations for variable x**
   3. Number and type of input and output parameters:

**using traversal of ast and linked list**

Type in Output: **expected and returned argument types do not match**

Number in Output: **expected and returned argument numbers do not match**

Type in input: **expected and passed argument types do not match**

Number in output: **expected and passed argument numbers do not match**

* 1. assignment of value to the output parameter in a function: **incompatible types for assignment**
  2. function call semantics:

**undefined reference to module func1** (checked function table)

**ERPLAG language does not support recursion** (should not call itself)

**redundant declaration and definition for module func1 before use in func2 module**

**(NOTE: when func1 is called by func2)**

* 1. static type checking :

**incompatible types for operands** (operands should be of same type)

**operations are not allowed on array constructs** (operations not allowed on arrays except a := b)

**invalid types for arithmetic operator** (should be numbers ie real/integers)

**invalid types for logical operator** (should be boolean values)

**invalid types for relational operator** (should be numbers)

**invalid index type for array** (array index has to be an integer)

**type mismatch, expected 'integer'** (or boolean/real as per the case)

**incompatible types for assignment** (lhs and rhs should have same type)

* 1. return semantics:

Type in Output: **expected and returned argument types do not match**

Number in Output: **expected and returned argument numbers do not match**

* 1. Recursion : **ERPLAG language does not support recursion** (should not call itself)
  2. module overloading: **multiple definitions for variable func1** (checked using hash table of functions)
  3. 'switch' semantics :

**real constructs are not allowed**

**boolean must have exactly two test cases: true and false and no default statement**

**integer must have a default statement and at least one case**

* 1. 'for' and 'while' loop semantics: **range and iterating variable in for loop can only be integer, range should be valid (begin<=end), while loop should have at least one condition inside loop that updates the while expression, while expression has to be a boolean**
  2. handling offsets for nested scopes: **symbol table contains an array of child pointers to nested symbol tables, starting offset of child[0] is the current offset of parent symbol table, similarly, starting offset of child[i] is current offset of child[i-1] (current offset means the end offset)**
  3. handling offsets for formal parameters: **offsets are stored in a separate symbol table contain only the formal parameters starting 0 (called InputSymbolTable)**
  4. handling shadowing due to a local variable declaration over input parameters:

**every reference to a local variable finds the occurrence first in the localSymbolTable. Only if it is not found there, it finds the element in inputSymbolTable (containing parameters), thus handling shadowing**

* 1. array semantics and type checking of array type variables: **if an array is static, range should be valid (begin<=end), if it is dynamic only type checking is done. In an expression A[i], i has to be an integer and A[i] has to be an integer for arithmetic expressions and boolean for logical expressions. Bound checking is done statically for static indices (A[5]) and dynamically for dynamic indices (A[i])**
  2. Scope of variables and their visibility : **Each variable is only visible inside the construct it has been declared and scope is limited to that (function/ for/ switch/ while), local variables shadows the input parameters if they have the same name, local variables shadow the variables of same name which were declared in the parent scope**
  3. computation of nesting depth:

**0: symbol table for input and output parameters**

**1 : symbol table that a function points to (localSymbolTable) (at the module level)**

**2: each child of symbol table 1 (at switch/loop level)**

**Thus, nesting depth of ST = nesting depth of parent ST + 1**

1. 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 = \_\_\_\_\_\_\_\_\_**2**\_\_\_\_\_\_\_\_\_\_\_(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):\_\_\_\_**rdi, rsi, rax for printf/scanf calls**\_\_\_\_\_\_**r15w for loops**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): \_\_\_\_\_\_\_\_\_\_**2**\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ **locations**), \_\_\_\_\_**2**\_\_\_\_\_\_\_\_\_\_(in **bytes**)

size(real): \_\_\_\_\_\_\_\_\_\_\_\_\_**4**\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ **locations**), \_\_\_\_\_**4**\_\_\_\_\_\_\_\_\_\_(in **bytes**)

size(boolean): \_\_\_\_\_\_\_\_\_\_**1**\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ **locations**), \_\_\_\_\_**1**\_\_\_\_\_\_\_\_\_\_(in **bytes**)

(**Assumption : Memory is byte organized)**

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

**input parameters are pushed on stack, thus on function call, stack has:**

**input n, input n-1, …, input1, return address (8 locations), rbp, local variables of called function**

**thus input parameters are referenced as locations [rbp+8+offset], similarly upon return, output parameters are pushed onto the stack and referenced using [rsp+offset]**

* 1. Specify the following:
     + Caller's responsibilities:

1. **push input parameters**
2. **readjust the stack to 16 locations boundary (required)**
3. **call function**
4. **pop the output parameters from stack**
5. **pop input paramters and adjust the stack**
   * + Callee's responsibilities:
6. **store rbp**
7. **allocate space for local variables and input parameters**
8. **adjust input parameters offsets to account for 8 bytes for return address in between**
9. **push output parameters**
10. **make sure return address is pushed above output parameters**
11. **deallocate space alloted for local variables (adjust rsp) and return**
    * + How did you maintain return addresses? (write 3-5 lines):

**when a function is called it looks like:**

**func1 space ,input params, return address, rbp, output params of func2, local variables of func2**

**thus, return address is always preserved on the stack**

* 1. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee?

**the callee uses offsets as calculated earlier for output and local variables as [rbp - offset] while for input parameters it needs to do [rbp + offset + 8]. Thus, offsets for input parameters are multiplied by -1 and subtracted with 8 to make it [rbp - (offset+8)] = [rbp - newoffset], thus, following our previous formula**

* 1. How is a dynamic array parameter receiving its ranges from the caller?

**array base and ranges are passed as input parameters, thus word[rbp+8], word[rbp+10] gives the ranges (8 to account for return adddress)**

* 1. What have you included in the activation record size computation? (local variables, parameters, both): \_\_\_\_\_\_\_\_\_\_\_**both local variables and parameters (input/output)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. register allocation (your manually selected heuristic) :

**used rax for calculating operands and stored result in rbx, rcx for counting, rbp as base pointer, rsp as stack pointer, rdi,rsi for printf/scanf calls and r15 for loops**

* 1. Which primitive data types have you handled in your code generation module?(Integer, real and boolean):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**integer and boolean only**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. Where are you placing the temporaries in the activation record of a function? \_\_\_\_\_**No, temporaries are never created. Direct code generation has been done**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Compilation Details**:
   1. Makefile works (yes/No):\_\_\_\_\_**Yes**\_\_\_\_\_\_
   2. Code Compiles (Yes/ No):\_\_\_\_\_\_**Yes**\_\_\_\_\_\_\_\_
   3. Mention the .c files that do not compile:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Yes**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. Any specific function that does not compile:\_\_\_\_\_\_\_\_\_\_\_\_\_**No**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   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 second = 1000000 ticks)**
   * 1. t1.txt (in ticks) \_\_\_\_\_\_\_**10729**\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_**0.010729**\_\_\_\_\_\_\_\_\_
     2. t2.txt (in ticks) \_\_\_\_\_\_\_\_**9885**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.009885**\_\_\_\_\_\_\_\_\_\_
     3. t3.txt (in ticks) \_\_\_\_\_\_\_\_**18789**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.018789**\_\_\_\_\_\_\_\_
     4. t4.txt (in ticks) \_\_\_\_\_\_\_\_\_**17261**\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_**0.017261**\_\_\_\_\_\_\_
     5. t5.txt (in ticks) \_\_\_\_\_\_\_\_\_**16341**\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_**0.016341**\_\_\_\_\_\_\_
     6. t6.txt (in ticks) \_\_\_\_\_\_\_\_**24863**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.024863**\_\_\_\_\_\_\_\_
     7. t7.txt (in ticks) \_\_\_\_\_\_\_\_**24153**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.024153**\_\_\_\_\_\_\_\_
     8. t8.txt (in ticks) \_\_\_\_\_\_\_**30264**\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_**0.030264**\_\_\_\_\_\_\_\_\_
     9. t9.txt (in ticks) \_\_\_\_\_\_\_\_**35097**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.035097**\_\_\_\_\_\_\_\_
     10. t10.txt (in ticks) \_\_\_\_\_\_\_\_**8482**\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_**0.008482**\_\_\_\_\_\_\_\_\_
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 dynamic arrays and real numbers**

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 -g -no-pie code.o**

**./a.out**

**The commands should be run stepwise**

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: **In case you get a segmentation fault with an option, please re-run that option again and it will perform the required tasks.** GCC has some portable issues when running huge programs (depending on the laptop architecture) and might give a segmentation fault at some times
3. Declaration: We, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_**Anirudh Goyal, Anish SS Kumar, Rohit Jain and Aditya Saxena** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (your names) 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: **2017A7PS0031P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Anirudh Goyal**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0069P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Anish SS Kumar**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0122P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name:  **Rohit Jain**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID: **2017A7PS0166P**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Name: **Aditya Saxena**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: **20 April 2020**

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

Should not exceed 6 pages.