**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: 2017A7PS0009P Name: Ritik Kandoria

ID: 2017A7PS0014P Name: Sanyam Jain

ID: 2017A7PS0155P Name: Deshmukh Advait Mahesh

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

1 lexerDef.h 7 parser.h 13 codegen.h 19\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 lexer.h 8 parser.c 14 codegen.c 20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 lexer.c 9 symboltable.h 15 driver.c 21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 getgrammar.h 10 symboltable.c 16 makefile 22\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 getgrammar.c 11 semanticAnalyzer.h 17 t1.txt-t10.txt 23\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 parserDef.h 12 semanticAnalyzer.c 18 c1.txt-c11.txt 24\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Total number of submitted files: 38 (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] YES
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 3 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 test cases (C#.txt, #:1-11): Not able to use NASM
   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): NO
   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 : This structure contains isTerminal ,complete information of terminal and non-terminal, pointer to children,pointer to parent,number of children, rule num of grammar used, index of child , generated
   2. Symbol Table structure: Hash of hash tables storing info such as name, numvalue, scope , type,offset,width, inputvar, r1 , r2, isdynamic, parent scope, startline, endline, isarray , depth level.
   3. Input parameters type structure: stored as string in symbol table
   4. Output parameters type structure: stored as string in symbol table
   5. Structure for maintaining the three address code(if 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 : Entry not found in symbol table in correct scope.
   2. Multiple declarations: Symbol table entry already found populated
   3. Number and type of input and output parameters: traversal of linked list of parameters and respective types is stored in symbol table in correct scope.
   4. assignment of value to the output parameter in a function: /////
   5. function call semantics: Checking function call definition found or not in symbol table. If found then matching input parameters and output parameters in the callee and the called function.
   6. static type checking : Type of left side is compared with left side of right side.
   7. return semantics: /////
   8. Recursion : Comparing the name of the function called and parent scope of call statement.
   9. 'switch' semantics :Type of switch variable is found and then default case is checked and then type of each case variable is compared with switch variable.
   10. 'for' and 'while' loop semantics: in for , the type of iterator is checked to be integer and in while the type of parameter is checked to be of type boolean.
   11. handling offsets for nested scopes: A count variable is entered into the symbol table.
   12. handling offsets for formal parameters:A variable keeps entry of offset in the symbol table.
   13. handling shadowing due to a local variable declaration over input parameters: Before declaring a variable it is checked its entry in the symbol table, if found then checked whether it is an input parameter or not.
   14. array semantics and type checking of array type variables:
   15. Scope of variables and their visibility : As soon as a scope starts, a global variable contains new scope and once a variable is declared it is entered into the correct scope table.
   16. computation of nesting depth: Making a tree of scopes and depth of node determines nesting depth.
2. Code Generation:
   1. NASM version as specified earlier used (Yes/no): Yes
   2. Used 32-bit or 64-bit representation: 32-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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* for stack pointer: esp
* others (specify): eax, ebx
  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), 4 (in bytes)

size(real): 4 (in words/ locations) 8 (in bytes)

size(boolean): 1 (in words/ locations), 2 (in bytes)

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

First the entry of function definition found or not is checked and then whether it is a recursive call or not is checked. If the function definition is found and the call is not recursive then input parameters and output parameters of caller’s function and callee function are compared. If they are found equal then function call is correct and no error is printed.

* 1. Specify the following:
     + Caller's responsibilities: First the entry of function definition found or not is checked and then whether it is a recursive call or not is checked. If the function definition is found and the call is not recursive then input parameters and output parameters of caller’s function and callee function are compared.
     + Callee's responsibilities: The input parameters can be reinitialised and output parameters must be assigned before returning.
  2. How did you maintain return addresses? (write 3-5 lines): Using activation record stack and putting entries on the top before a call is made and space for new activation is reserved. and then an entry is made in the new space where to return once the return statement is executed and then new space is deleted.
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? In type checking yes we have maintained parameter passing .
  4. What have you included in the activation record size computation? (local variables, parameters, both): Both
  5. register allocation (your manually selected heuristic) : ////s
  6. Which primitive data types have you handled in your code generation module?(Integer, real and boolean):Integer
  7. Where are you placing the temporaries in the activation record of a function? ////

1. **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: 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. t1.txt (in ticks) 15625 and (in seconds) 0.015625
     2. t2.txt (in ticks) 15628 and (in seconds) 0.015628
     3. t3.txt (in ticks) 16530 and (in seconds) 0.016530
     4. t4.txt (in ticks) 15728 and (in seconds) 0.015728
     5. t5.txt (in ticks) 34340 and (in seconds) 0.034340
     6. t6.txt (in ticks) 2454 and (in seconds) 0.02454
     7. t7.txt (in ticks) 15770 and (in seconds) 0.015770
     8. t8.txt (in ticks) 44225 and (in seconds) 0.044225
     9. t9.txt (in ticks) 54340 and (in seconds) 0.054340
     10. t10.txt (in ticks) 54311 and (in seconds) 0.054311
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)

Dynamic arrays and function call at run time. ///

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 elf code.asm

$ ld -m elf\_i386 -s -o code code.o -lc

$ ./code

1. **Strength of your code**(Strike off where not applicable): (a) correctness (c) robustness (e) readable (f) strong data structure (f) Good programming style (indentation, avoidance of goto stmts etc) (g) modular
2. Any other point you wish to mention: No

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

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

1. Declaration: We, Ritik Kandoria, Sanyam Jain, Deshmukh Advait Mahesh 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: 2017A7PS0009P Name: Ritik Kandoria

ID: 2017A7PS0014P Name: Sanyam Jain

ID: 2017A7PS0155P Name: Deshmukh Advait Mahesh

Date:20-04-2020

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

Should not exceed 6 pages.