**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: 2016B3A70398P Name: Ayush Vachaspati

ID: 2016B1A70938P Name: Indraneel Ghosh

ID: 2016B1A70929P Name: G Adityan

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

1 lexerDef.h 7 ast.h 13 codegen.c 20 driver.c

2 parseDef.h 8 typechecker.h 14 parser.c 21 coding\_Details(stage 2).docx

3 astdef.h 9 symboltable.h 15 ast.c 22 makefile

4 symbolTableDef.h 10 semantic.h 16 typechecker.c 23 nary\_tree.h

5 lexer.h 11 codegen.h 17 symboltable.c 24 hashtable.h

6 parser.h 12 productions.txt 18 semantic.c 25 hashtable.c

26 hash.c 32 treefiles.c 38 t2.txt 44 t8.txt

27 nary\_tree.c 33 treefiles.h 39 t3.txt 45 t9.txt

28 color.c 34 color.h 40 t4.txt 46 t10.txt

29 stack.c 35 hash.h 41 t5.txt 47 c1.txt

30 set.h 36 set.c 42 t6.txt 48 c2.txt

31 stack.h 37 t1.txt 43 t7.txt 49 c3.txt

50 c4.txt 52 c6.txt 54 c8.txt 56 c10.txt

51 c5.txt 53 c7.txt 55 c9.txt 57 c11.txt 58 lexer.c

1. Total number of submitted files: 58 (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 \_\_\_\_ 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): \_Yes\_
   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):\_NA\_
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure\_{lexeme;datatype;token;ASTnode\*\* children; number\_of\_children; casehandle;}
   2. Symbol Table structure: The symbol table has a pointer to parent node and children nodes with relevant attributes required.
   3. array type expression structure: \_<isarray,isdynamic,type,range1,range2>\_
   4. Input parameters type structure: \_
   5. Output parameters type structure: \_
   6. Structure for maintaining the three address code(if created) : \_NA\_
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 : \_no entry in symbol table\_
   2. Multiple declarations: \_already has entry in symbol table\_
   3. Number and type of input and output parameters: \_function prototype stores and matched against call
   4. assignment of value to the output parameter in a function \_copy back from callee stack\_
   5. function call semantics: \_copy input parameters to callee and copy back output to caller\_
   6. static type checking : \_assign type to variable by traversing ast and check types of expressions\_
   7. return semantics: \_copy values back to caller \_
   8. Recursion : \_stack frame created for each call. Function cannot call itself \_
   9. module overloading: \_not allowed \_
   10. 'switch' semantics : \_checked that case values are of same type and that default is present for integer and not for boolean \_
   11. 'for' and 'while' loop semantics: \_loop semantics checked according to language spec\_
   12. handling offsets for nested scopes: \_hierarchy of symbol tables with parent and child pointers \_
   13. handling offsets for formal parameters: \_start from zero in stack frame \_
   14. handling shadowing due to a local variable declaration over input parameters: \_hierarchy of symbol tables, search from bottom up\_
   15. array semantics and type checking of array type variables: \_match range and array type\_
   16. Scope of variables and their visibility : \_using symbol table hierarchy\_
   17. computation of nesting depth: \_depth of symbol table tree node \_
8. Code Generation:
9. NASM version as specified earlier used (Yes/no): \_Yes\_
10. Used 32-bit or 64-bit representation: \_32-bit\_
11. For your implementation: 1 memory word = \_4\_ (in bytes)
12. Mention the names of major registers used by your code generator:

* For base address of an activation record: \_EBP\_
* for stack pointer: \_ESP\_
* others (specify): \_EAX,EBX,ECX,EDX\_
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): \_\_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_(in

bytes)

size(real): \_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_(in

bytes)

size(booelan): \_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_(in

bytes)

* 1. How did you implement functions calls?(write 3-5 lines describing your model of implementation) Caller copies the parameters into the calle stack fram and sets up the stack frame. Then ebp is pushed onto stack and then ebp is changed to esp. Then returning caller copies back the output variables and then ebp is restored.\_ Specify the following:
     + Caller's responsibilities: \_setup callee stack frame \_
     + Callee's responsibilities: \_computation and restore registers\_
  2. How did you maintain return addresses? (write 3-5 lines): \_Callers responsibility to copy back output. Caller has return addresses in its stack frame\_
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? \_Parameter passing is done by copying values from caller to callee. Callee uses statically computed offsets to access vars from its stack\_
  4. How is a dynamic array parameter receiving its ranges from the caller? \_ranges are maintained as first 2 words in the array\_
  5. What have you included in the activation record size computation? (local variables, parameters, both): \_both local variables and parameters\_
  6. register allocation (your manually selected heuristic) : \_ESP for stack pointer, EBP for base pointeredx, eax for computation and ecx for loops, edi esi to access arrays\_
  7. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): \_integer and boolean\_
  8. Where are you placing the temporaries in the activation record of a function? \_Stack\_

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 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) \_3480\_ and (in seconds) \_0.0348\_
     2. t2.txt (in ticks) \_5207\_ and (in seconds) \_0.05207\_
     3. t3.txt (in ticks) \_5332\_ and (in seconds) \_0.05332\_
     4. t4.txt (in ticks) \_5898\_ and (in seconds) \_0.05898\_
     5. t5.txt (in ticks) \_6675\_ and (in seconds) \_0.06675\_
     6. t6.txt (in ticks) \_7422\_ and (in seconds) \_0.07422\_
     7. t7.txt (in ticks) \_7545\_ and (in seconds) \_0.07545\_
     8. t8.txt (in ticks) \_7743\_ and (in seconds) \_0.07743\_
     9. t9.txt (in ticks) \_8112\_ and (in seconds) \_0.08112\_
     10. t10.txt (in ticks) \_8637\_ and (in seconds) \_0.08637\_
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)\_NA\_
5. Are you availing the lifeline (Yes/No): \_Yes\_
6. 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]

1. ./compiler <inputfile> <codefile.asm>

2. nasm -f elf -g <code\_filename.asm>

3. gcc -no-pie -m32 code.o -o <output\_filename>

4. ./<output\_filename>

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

(g) Good programming style (indentation, avoidance of goto stmts etc)

(h) modular

(i) space and time efficient

1. Any other point you wish to mention: \_NA\_
2. Declaration: We,Ayush Vachaspati, Indraneel Ghosh and G. Adityan (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: 2016B3A70398P Name: Ayush Vachaspati

ID: 2016B1A70938P Name: Indraneel Ghosh

ID: 2016B1A70929P Name: G Adityan

Date: \_21/04/2020\_

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

Should not exceed 6 pages.