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

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**II Semester 2022-23**

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

**Coding Details**

**(April 12, 2023)**

**Group number \_\_\_\_\_30\_\_\_\_\_(Write your group number here)**

*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:  **2020A7PS0094P**  Name**: Arya Veer Singh Chauhan**

ID:  **2020A7PS0016P**  Name**: Ruchika Sarkar**

ID:  **2020A7PS0984P**  Name**: Utsav Goel**

ID:  **2020A7PS0049P**  Name**: Madhav Madhusoodanan**

ID:  **2020A7PS0102P**  Name**: Hardik Jain**

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

1\_ constants.c 21\_ t2.txt 41\_ codeGen.h

2\_ constants.h 22\_ t3.txt 42\_ DFA\_IMAGE\_GROUP30.pdf

3\_ driver.c 23\_ t4.txt 43\_ ast\_group30.pdf

4\_ grammar.txt 24\_ t5.txt 44\_ First And Follow Set Group30.pdf

5\_ lexer.c 25\_ t6.txt 45\_ c1.txt

6\_ lexer.h 26\_ t7.txt 46\_ c2.txt

7\_ lexerDef.h 27\_ t8.txt 47\_ c3.txt

8\_ makefile 28\_ t9.txt 48\_ c4.txt

9\_ parser.c 29\_ t10.txt 49\_ c5.txt

10\_ parser.h 30\_ ast.c 50\_ c6.txt

11\_ parserDef.h 31\_ ast.h 51\_ c7.txt

12\_ testcase1.txt 32\_ c11.txt 52\_ c8.txt

13\_ testcase2.txt 33\_ astDef.h 53\_ c9.txt

14\_ testcase3.txt 34\_ symbolTable.c 54\_ c10.txt

15\_ testcase4.txt 35\_ symbolTable.h

16\_ testcase5.txt 36\_ symbolTableDef.h

17\_ testcase6.txt 37\_ intermediateCodeGen.c

18\_ testcase0.txt 38\_ intermediateCodeGen.h

19\_ coding\_Details(stage2) 39\_ intermediateCodeGenDef.h

20\_ t1.txt 40\_ codeGen.c

1. Total number of submitted files: **54** (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(Partial)**
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): **c1**

(We are unable to print and scan input from user in various types as we were unable to use gcc functions for print and scan, all the computation is working fine, eg for loop and other things but we implemented a partial print and scan function that works well if input and output are in the range of ‘0’-’9’ otherwise it displays ascii characters corresponding to the answer)

* 1. Semantic Analyzer produces semantic errors appropriately (Yes/No): **Yes**
  2. Static Type Checker reports type mismatch errors appropriately (Yes/ No): **Yes**
  3. Dynamic type checking works for arrays and reports errors on executing code.asm (yes/no): **No**
  4. Symbol Table is constructed (yes/no) **Yes**  and printed appropriately (Yes /No): **Yes**
  5. AST is constructed (yes/ no) **Yes** and printed (yes/no) **Yes**
  6. 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): **c2-c11**

1. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure: **Left child, Right child, list address syn, type, lexeme, flag.**
   2. Symbol Table structure: **Contains symbol table row(array range, ID, type, next, offset, params).**
   3. array type expression structure: **Type, left, right, leftSign, rightSign, offset.**
   4. Input parameters type structure: **Linked list of Symbol table rows.**
   5. Output parameters type structure: **Linked list of Symbol table rows.**
   6. Structure for maintaining the three address code(if created) : **Operator, operands, function associated, and data.**
2. **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 : **Symbol Table entry empty.**
   2. Multiple declarations: **Symbol table entry already found.**
   3. Number and type of input and output parameters: **Traversal of linked list of parameters and respective types.**
   4. assignment of value to the output parameter in a function: **Marking assignment with flag; checking flag at the end of function.**
   5. function call semantics: **Global symbol table, input output param check.**
   6. static type checking : **Type of AST node as well as in symbol table entry is compared.**
   7. return semantics: **Global symbol table, output param check.**
   8. Recursion : **Global current function, name checking.**
   9. module overloading: **Symbol table entry already found.**
   10. 'switch' semantics : **Type of AST node.**
   11. 'for' and 'while' loop semantics: **While: flagging each while loop expression variable in case of assignment inside while loop. For: flagging for loop variable in case of assignment or declaration.**
   12. handling offsets for nested scopes: **Every symbol table has a parent symbol table pointer and every symbol table has a base pointer.**
   13. handling offsets for formal parameters: **Base pointer of symbol table, offset corresponding to the types.**
   14. handling shadowing due to a local variable declaration over input parameters: **First we check local variable if not found then we check input parameters.**
   15. array semantics and type checking of array type variables: **Type of node and matching it with symbol table entry type.**
   16. Scope of variables and their visibility : **Found in symbol table or recursively in parent.**
   17. computation of nesting depth: **Recursively.**
3. Code Generation:
   1. NASM version as specified earlier used (Yes/no): **Yes**
   2. Used 32-bit or 64-bit representation: **64-bit representation**
   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: **origin**
* for stack pointer: **ESP**
* others (specify): NA
  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(booelan): **1** (in words/ locations), **1** (in bytes)

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

**For a function call , we naked starting of function and copied starting and ending values of loop into 2 registers and the subtracted to find the count , the number of times the loop will work and stored in it cx register. Then we pushed all statements in between and marked end of for\_loop. After it we started decreasing cx in every iteration and checked if it turned out 0 if not we jumped to statements again.**

* 1. Specify the following:
     + Caller's responsibilities: **Caller PUSH every value of input\_param into memory stack**
     + Callee's responsibilities: **It POPS every value in given order and store it in location assigned to variable of input\_param.**
  2. How did you maintain return addresses? (write 3-5 lines): **To maintain base address, we push current state of system into memory stack(by using push EBP and then POP it just before returning to function.**
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? **solved in intermediate code generation.\_**
  4. How is a dynamic array parameter receiving its ranges from the caller? **Not working**
  5. What have you included in the activation record size computation? (local variables, parameters, both): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**local variables**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  6. register allocation (your manually selected heuristic) : **Not Applicable**
  7. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): **ALL**
  8. Where are you placing the temporaries in the activation record of a function?

**Handled in intermediate code generation by making temporary variables and assigning values of location to them**

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) 1826 and (in seconds) 0.001826
     2. t2.txt (in ticks) 5285 and (in seconds) 0.005285
     3. t3.txt (in ticks) 6807 and (in seconds) 0.006807
     4. t4.txt (in ticks) 6590 and (in seconds) 0.006590
     5. t5.txt (in ticks) 6977 and (in seconds) 0.006977
     6. t6.txt (in ticks) 5430 and (in seconds) 0.005430
     7. t7.txt (in ticks) 6987 and (in seconds) 0.006987
     8. t8.txt (in ticks) 7716 and (in seconds) 0.007716
     9. t9.txt (in ticks) 13541 and (in seconds) 0.013541
     10. t10.txt (in ticks) 6248 and (in seconds) 0.006248
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 type checking, Dynamic bound checking in case of arrays, Function dynamic type checking.

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 -f elf64 finalCode.asm && ld finalCode.o -o finalCode && ./finalCode**

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: Due to memory efficiency we have exited from the driver function after every command. So please check everything separately.
3. Declaration: We, Arya Veer Singh Chauhan, Madhav Madhusoodanan, Ruchika Sarkar, Utsav Goel, Hardik Jain (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: 2020A7PS0094P Name: Arya Veer Singh Chauhan

ID: 2020A7PS0049P Name: Madhav Madhusoodanan

ID: 2020A7PS0016P Name: Ruchika Sarkar

ID: 2020A7PS0984P Name: Utsav Goel

ID: 2020A7PS0102P Name: Hardik Jain

Date: 02-03-2023 Group number: 30

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

Should not exceed 6 pages.