**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: 2016B5A70590P Name: Anwesh Bhattacharya

ID: 2017A7PS0147P Name: Deepak Chahar

ID: 2017A7PS0103P Name: Komal Vasudeva

ID: 2016B1A70822P Name: Rohan Kela

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

1 parser.c 7 lexerDef.h 13 codegen.c

2 lexer.c 8 makefile 14 SymbolTable.c

3 lexer.h 9 lexTest.c 15 SymbolTable.h

4 parser.h 10 driver.c 16 typeChecker.c

5 parserDef.h 11 ast.c 17 typeChecker.h

6 grammmar.txt 12 ast.h 18

1. **Total number of submitted files:** 54
2. **Have you mentioned names and IDs of all team members at the top of each file** YES
3. **Have you compressed the folder as specified in the submission guidelines?** YES
4. **Status of Code development**: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
   1. **Lexer** : Yes
   2. **Parser** : Yes
   3. **Abstract Syntax tree** : Yes
   4. **Symbol Table** : Yes
   5. **Type checking Module** : YES
   6. **Semantic Analysis Module** : YES (reached LEVEL 4 as per the details uploaded)
   7. **Code Generator** : YES
5. **Execution Status**:
   1. **Code generator produces code.asm** : 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 **and printed appropriately : YES**
   7. **AST is constructed**  YES **and printed** 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**: Pointers to store next, child & parent. Pointer to store local symbol table. Pointer to store data\_type, token information and tokenId wherever applicable.
   2. **Symbol Table structure:**List of local, input, output variables, activation record size and
   3. **array type expression structure**: Structure with base type, and limits
   4. **Input parameters type structure**: offset, name, base type, array limits if any
   5. **Output parameters type structure**: offset, name, type
   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** : Checked with symbol table entry
   2. **Multiple declarations**: Consulting symbol table
   3. **Number and type of input and output parameters** : Matching corresponding formal and actual arguments with symbol tables of caller and callee
   4. **assignment of value to the output parameter in a function** Checked with symbol table entry
   5. **function call semantics**: Checked
   6. **static type checking** : Recursive function on the expression tree with the help of symbol table
   7. **return semantics**: Matched corresponding symbol table entries of caller and callee
   8. **Recursion** : Checked with symbol table entry
   9. **module overloading**: checked with symbol table entry
   10. **'switch' semantics** : Check type of switch variable from symbol table and decide whether default is required or not
   11. **'for' and 'while' loop semantics**: Checked for modification of loop variables by symbol table entry
   12. **handling offsets for nested scopes**: Offset has been calculated relative to a variable’s parent most function
   13. **handling offsets for formal parameter**s: Maintain depth of current call stack frame
   14. **handling shadowing due to a local variable declaration over input parameters**: Search symbol table
   15. **array semantics and type checking of array type variables**: Type checking of array index done atcompile time, static bound check done at compile time
   16. **Scope of variables and their visibility** : Implemented by recursive hiding
   17. **computation of nesting depth**: Recusive function to calculate it
8. Code Generation:
   1. NASM version as specified earlier used : YES
   2. Used 32-bit or 64-bit representation: **64**
   3. For your implementation: 1 memory word = 1 byte
   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, RBX, RCX, RDX, RDI, RSI
  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): \_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_(in bytes)

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

At first, the RSP is subtracted by the suitable amount to provide space for the return variables. Then the formal arguments are pushed into the stack in reverse order. The caller is called using **call** instruction. Hence input and output variables are accessed in the caller above the base pointer, and local variables are accessed below the base pointer. Then after the **ret** instruction from the caller, the control passes back to the callee and it receives the return variables by appropriately popping

* 1. Specify the following:
     + Caller's responsibilities: Reserve space for return variables and push the formal arguments onto the stack
     + Callee's responsibilities: Access the appropriate input/output variables **above** the base pointer an then assign the appriopriate values to the return variables.
  2. How did you maintain return addresses? (write 3-5 lines): Return addresses is automatically maintained by **call** and **ret** instruction
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? Pointer to the static array along is passed
  4. How is a dynamic array parameter receiving its ranges from the caller? Dynamic array pointer is passed
  5. What have you included in the activation record size computation? (local variables, parameters, both): Local variables and output parameters
  6. register allocation (your manually selected heuristic) : AX > BX > DX. CX for counters and loops
  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? There are no temporaries in our implementation

1. **Compilation Details**:
   1. Makefile works : Yes
   2. Code Compiles : Yes
   3. Mention the .c files that do not compile: All compile
   4. Any specific function that does not compile: All compile
   5. Ensured the compatibility of your code with the specified versions [GCC, UBUNTU, NASM] L 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) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 2.5 ms
     2. t2.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 2.6 ms
     3. t3.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 2.7 ms
     4. t4.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 2.0 ms
     5. t5.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 2.5 ms
     6. t6.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 3 ms
     7. t7.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 4.5 ms
     8. t8.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 5ms
     9. t9.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 6ms
     10. t10.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) 1.5 ms
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)

Passing of dynamic arrays in function calls

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 -felf64 code.asm*

*> gcc -no-pie code.o*

*> ./a.out*

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:

* Unary expressions are not handled
* Array assignment is not handled
* Type checking of array index was done at compile time
* We've had to constantly monitor the stack pointer location in order to take care of 16 byte alignment of scanf
* Functions may/may not be handled. Depends on my progress.

1. Declaration: We, Rohan Kela, Anwesh Bhattacharya, Deepak Chahar, Komal Vasudeva (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: 2016B5A70590P Name: Anwesh Bhattacharya

ID: 2017A7PS0147P Name: Deepak Chahar

ID: 2017A7PS0103P Name: Komal Vasudeva

ID: 2016B1A70822P Name: Rohan Kela

**Date: 20-04-2020**