BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

Group Number
14

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:

ID: 2016B2A70773P **Name: Ishan Sharma**

ID: 2017A7PS0152P **Name: Sanjeev Singla**

ID: 2015B5A70749P **Name: Sarthak Sahu**

ID:2017A7PS0142P **Name: Anirudh Garg**

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

1. ast.c 2. ast.h 3. c1.txt 4. c2.txt 5. c3.txt 6. c4.txt 7. c5.txt
8. c6.txt 9. CodeGen.c 10. CodeGen.h 11. driver.c 12. first.txt 13. follow.txt
14. grammar.txt 15. lexer.c 16. lexer.h 17. lexerDef.h 18. makefile 19. parser.c
20. parser.h 21. parserDef.h 22. SymbolTable.c 23. SymbolTable.h 24. SymbolTableDef.h
25. t1.txt 26. t2.txt 27. t3.txt 28. t4.txt 29. t5.txt 30. t6.txt 31. t7.txt
32. t8.txt 33. t9.txt 34. t10.txt 35. TypeChecker.c 36. TypeChecker.h
37. Coding_Details_Group_14

Type Checker and Semantic Analysis have been integrated together in files TypeChecker.c and TypeChecker.h.

3. Total number of submitted files: **37** (All files should be in **ONE** folder named exactly as Group number)
4. 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]
5. Have you compressed the folder as specified in the submission guidelines? (yes/no) **YES**
6. **Status of Code development**: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
    a. Lexer (Yes/No): **Yes**
    b. Parser (Yes/No): **Yes**
    c. Abstract Syntax tree (Yes/No): **Yes**
    d. Symbol Table (Yes/ No): **Yes**
    e. Type checking Module (Yes/No): **Yes**
    f. Semantic Analysis Module (Yes/ no): Yes (reached LEVEL **4** as per the details uploaded)
    g. Code Generator (Yes/No): **YES**

7. **Execution Status**:
    a. Code generator produces code.asm (Yes/ No): **Yes**
    b. code.asm produces correct output using NASM for testcases (C#.txt, #:1-11): **c1-c6.txt**
    c. Semantic Analyzer produces semantic errors appropriately (Yes/No): **Yes**
    d. Static Type Checker reports type mismatch errors appropriately (Yes/ No): **Yes**
    e. Dynamic type checking works for arrays and reports errors on executing code.asm (yes/no): **No**

       Dynamic Arrays have not been implemented but, dynamic index for static arrays has been implemented.
    f. Symbol Table is constructed (yes/no) Yes and printed appropriately (Yes /No): **Yes**
    g. AST is constructed (yes/ no) Yes and printed (yes/no) **Yes**
    h. 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): **t1-t10.txt & c1-c6.txt – NONE, cannot handle c7-c11.txt**
8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)

    a. AST node structure : **Not creating a new structure but making changes in the Parse Tree Structure.**

       Parse Tree Structure: **n-ary Tree having, link to right,left,parent,child node.**

       Link to its corresponding SymbolTable entry and scope if it is an ID.

       Type Field for Type Checking and Semantic Analysis.

       isCG (flag set if code is generated for that node) and temporary (stores temporary for ID) for CodeGen Phase.
    b. Symbol Table structure: **n-ary Tree, each node is a new scope and has linklist (nodehead) of scope variable entries**.

       Each entry is a different struct as SymbolEntry
    c. array type expression structure: **Structure having isDynamic, ifnumvalue, id, isused; SymbolTable entry**
    d. Input parameters type structure: **Node of Symbol Table n-ary Tree**
    e. Output parameters type structure: **Node of Symbol Table n-ary Tree**
    f. Structure for maintaining the three address code(if created) **N.A**

9. **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.]
    a. Variable not Declared **: Symbol Table entry not found**
    b. Multiple declarations: **Symbol Table entry already exists**
    c. Number and type of input and output parameters: **Comparing formal and actual parameters by comparing their types in their respective symbol table entries and also counting the number while traversal**
    d. assignment of value to the output parameter in a function: **Traversing of whole function, assigning 1 to isAss variable in SymbolTable Entry Structure is assigned.**
    e. function call semantics: **Using SymbolTable 'global' entry. 'global' symbol table contains all the function entries as linklist.**

f. static type checking : **Left Child, Right Child type comparison for an operator**

g. return semantics: **Comparing formal and actual parameters by comparing their types in their respective symbol table entries and also counting the number while traversal** _

h. Recursion : **moduleReuseStmt ID shouldn't match the function name in which the stmt is in.**

i. module overloading: **ID of new function definition shouldn't match with any other previously defined functions.**

j. 'switch' semantics : **Checking if ID of switch is of type Integer or Boolean (in its symbol table entry type)**

if type = real then error.

If ID type = integer, traverse the AST to find default, if not present report error else no error.

If ID type = Boolean, traverse the AST to check if default present, if present report error else no error.

k. 'for' and 'while' loop semantics:

Traversing all statements inside 'for' and setting flag isAss = 1 if the iterating variable is redefined in the for loop, if ultimately isAss == 1 ERROR, else no error.

Traversing all statements inside 'while' and setting flag isAss = 1 If the variables participating in while expression appear in any value assignment statement. Ultimately, if even 1 variable is assigned inside while loop -> no error. If no variable was assigned inside while loop -> report error.

l. handling offsets for nested scopes**: Offset set to 0 for every moduleDef start.**

m. handling offsets for formal parameters: **Offset set to 0 for input_plist statement and the same offset continues for output_plist variables.**

n. handling shadowing due to a local variable declaration over input parameters: **Local Variables are stored in function symbol table, and input_plist parameters are stored in input_plist symboltable which the leftmost child of the aforementioned function symboltable.**

o. array semantics and type checking of array type variables: **All type checking, index checking etc has been done using traversal of AST and corresponding array symbol table entry**

p. Scope of variables and their visibility: **Symbol Table Entry**

q. computation of nesting depth: **Traversal of symbol table n-ary tree structure.**

10. Code Generation:

a. NASM version as specified earlier used (Yes/no): **Yes**
b. Used 32-bit or 64-bit representation: **64 bit**
c. For your implementation: 1 memory word = **4** (in bytes)
d. Mention the names of major registers used by your code generator:
   - For base address of an activation record: **N.A**
   - for stack pointer: **N.A.**
   - others (specify): **N.A.**

e. Mention the physical sizes of the integer, real and boolean data as used in your code generation module
size(integer): 1 (locations), 4 (in bytes)
size(real): 1 (locations), 4 (in bytes)
size(booelan): 1 (locations), 4 (in bytes)

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

We couldn't implement.

g. Specify the following:

- Caller's responsibilities: **N.A**
- Callee's responsibilities: **N.A**

h. How did you maintain return addresses? (write 3-5 lines): **N.A**

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

j. How is a dynamic array parameter receiving its ranges from the caller? **N.A**

k. What have you included in the activation record size computation? (local variables, parameters, both): **Local Variables only**

l. register allocation (your manually selected heuristic) : **N.A**

m. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): **integer, boolean**

n. Where are you placing the temporaries in the activation record of a function? **N.A**

11. **Compilation Details**:

a. Makefile works (yes/No): **Yes**

b. Code Compiles (Yes/ No): **Yes**

c. Mention the .c files that do not compile: **N.A**

d. Any specific function that does not compile: **N.A**

e. Ensured the compatibility of your code with the specified versions [GCC, UBUNTU, NASM] (yes/no)

Yes

But the code.asm files for c1-c6.txt had been executed and checked only on a Ubuntu Linux Virtual Machine on a MacBook because that is the only linux machine we had in the group. There were no segmentation faults for c1-c6.txt when we checked on the virtual machine and were working as expected. Ma'am, if you encounter segmentation faults, please contact Ishan Sharma.

12. Execution time for compiling the test cases [lexical, syntax and semantic analyses including symbol table creation, type checking and code generation] :

i. t1.txt (in ticks) **12822** and (in seconds) **0.012822s**

ii. t2.txt (in ticks) **8973** and (in seconds) **0.008973s**

iii. t3.txt (in ticks) **37355** and (in seconds) **0.037355s**

iv. t4.txt (in ticks) **25236** and (in seconds) **0.025236s**

v. t5.txt (in ticks) **36099** and (in seconds) **0.036099s**

vi. t6.txt (in ticks) **64724** and (in seconds) **0.064724s**

vii. t7.txt (in ticks) **74510** and (in seconds) **0.074510s**

viii. t8.txt (in ticks) **89903** and (in seconds) **0.089903s**

ix. t9.txt (in ticks) **148507** and (in seconds) **0.148507s**

x. t10.txt (in ticks) **12434** and (in seconds) **0.012434s**

13. **Driver Details**: Does it take care of the **TEN** options specified earlier?(yes/no): **Yes**

14. Specify the language features your compiler is not able to handle (in maximum one line)

    Lexer to Semantic Analyser can handle all language features.

    But CodeGen can handle only driver module with static arrays.

15. Are you availing the lifeline (Yes/No): **No**
16. 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 -o code.o code.asm

    gcc -no-pie code.o -o code

    ./code

17. **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

18. Any other point you wish to mention: **N.A**
19. Declaration: We, **Ishan Sharma, Sarthak Sahu, Sanjeev Singla, Anirudh Garg** (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: 2016B2A70773P	**Name: Ishan Sharma**
ID: 2017A7PS0152P	**Name: Sanjeev Singla**
ID: 2015B5A70749P	**Name: Sarthak Sahu**
ID:2017A7PS0142P	**Name: Anirudh Garg**

Date: 20 April, 2020

Should not exceed 6 pages.