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 __________29_______________(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:__2019B3A70459P______________Name:_____Shaurya_Marwah___________________________________

   ID:__2019B5A70650P______________Name:_____Ruchir_Kumbhare__________________________________

   ID:__2019B2A70957P______________Name:_____Ashwin_Murali_____________________________________

   ID:__2019B3A70564P______________Name:_____Hari_Sankar_______________________________________

   ID:__2020A7PS1203P______________Name:_____Dilip_Venkatesh___________________________________

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

   1_____driver.c___________ 7______parser.h______________ 13____hashmap.h____________
   2______grammar.txt______ 8_____parser.c_______________ 14________makefile________
   3______lexerDef.h__________9______twinbuffer.h_____________ 15_____nonterms.txt_______
   4________lexer.c__________ 10____testcases ________________ 16___parseTree.txt_____________
   5_______lexer.h___________11_____Coding Details pro forma_ 17_______set.c_________
   6_________parseDef.h_____ 12_________hashmap.c__________ 18____set.h_____________
   19. stack.c 20. stack.h 21. token_name.h 22. tokens.txt 23.tree.c 24. tree.h 25. twinbuffer.c
   26. ast.c 27. ast.h 28. astDef.h 29. astLabels.txt 30.codeGenDef.h 31.codegen.c
   32. intermCodeGenDef.h 33.intermedCodeGen.c 34. intermedCodeGen.h 35. symTableUtil.c
   36. symTableUtil.h 37. symbolTable.c 38. symbolTable.h 39. symbolTableDef.h
   40.typechecker.c 41. typecheckerdef.h 42.Grammar_firstandfollow.pdf 43.AST.pdf 44. DFA.pdf

3. Total number of submitted files: ______44_____ (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.txt, C2.txt,C3.txt,C4.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_____________
    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):_______c5.txt, c6.txt, c7.txt,c8.txt, c9.txt,c10.txt, c11.txt(we have not implemented code generation for constructs mentioned in these testcases)___________________________________

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
    a. AST node structure - **is modelled in a left child and siblings representation, each node has a label associated (enumerated), a symbol table pointer, and some attributes like code, name, truecase, falseCase (for intermediate codegen) and attributes for line start and line end**
    b. Symbol Table structure - **has first a union of typeIfNotArray and typeIfArray, the input and output param list, the corresponding hashtable, parent hashtable and sibling hashtables, in addition to this some attributes to keep the offsets and width of the variables, activation record of function etc.**
    c. array type expression structure: **has integers low and high (which contain the ranges if static), lowLexeme and highLexme (if the ranges are dynamic), and boolean vars to indicate if the dynamic ranges have a negative sign associated with them. In addition to this two additional booleans are kept to check which bounds are statically available**
    d. Input parameters type structure: **Modelled as a linkedlist with attributes as typeIfArray or typeIfNotArray, and width, offset, and a boolean indicating whether this parameter is an array or not, a next pointer is also kept**
    e. Output parameters type structure:**Same as the input parameter type structure**
    f. Structure for maintaining the three address code(if created) : **Modelled a quadruple with operands information (2 attributes), and resultant information (all are kept as fixed size strings, not pointers). In addition to this, the quadruple also contains offset information (if it exists) and also the corresponding the type expression (which may be required during code generation phase).**

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 empty**
    b. Multiple declarations: **Symbol table has an existing entry for the variable or module.**
    c. Number and type of input and output parameters: **traverse the linkedlist of input and output parameters and compare types and number of parameters.**
    d. assignment of value to the output parameter in a function: **Check all the assignment statements and module reuse statements (if output parameters exist for the function call), within the local scope of**

the module and if we encounter switch case, for stmt, or while, need to check the statements inside those too, did this for every output parameter to check if was assigned a value**

e. function call semantics: **needed to check first if the module which was called existed or not, once that is done, need to check the if the types of the actual and formal parameters was same as the function definition.**

f. static type checking : **according to the type checking rules, propagated an attribute type from the bottom towards up in the ast, after that just check if type_error is propagated up or not.**

g. return semantics: **The number and type of output parameters would be compared (optional) with the formal parameters present in the function definition**

h. Recursion : **Get the module's symbol table (inside where we are currently), and check if the function call has the same function name.**

i. module overloading: **Check if the global symbol table has a pre-existing entry for the module with the same name.**

j. 'switch' semantics : **determined the type of the switch label (which cannot be real), if it was boolean checked if default node was non-empty(error) and if it were integer, checked if default node was absent(error)**

k. 'for' and 'while' loop semantics: **For every statement in the for construct, checked if the loop variable was not assigned any value (assignment and module reuse statements), for while, checked the statements again and again for every variable that appears in its condition (if atleast one is assigned, returned true)**

l. handling offsets for nested scopes: **The offsets are assigned in an incremental way, as and when a new scope is encountered, the offset at that point is passed to the new scope, after offset computation is done, the offsets are resumed from the same place (as were towards the last statement in the scope).**

m. handling offsets for formal parameters: **the offsets start from zero for any module, first input and output parameters are assigned offsets, the next offset is given to the statements inside the module.**

n. handling shadowing due to a local variable declaration over input parameters: f**or every declaration, need to check if it is an input parameter or not, if it is, redeclaration error is not raised.**

o. array semantics and type checking of array type variables: **the array element can contain expressions, so we have checked if the type for the expression is evaluating to integer or not, while type checking the constructs, static bound checking was done.**

p. Scope of variables and their visibility : **Nested hashtables, a child hashtable is essentially a new scope**

q. computation of nesting depth: **Since we have a tree like structure for hashtables, computing depth (distance from root to that node) of any node (which is a hashtable), would give us the nesting depth**

10. Code Generation:

a. NASM version as specified earlier used (Yes/no):__________Yes_______________
b. Used 32-bit or 64-bit representation:___________64___________________________________
c. For your implementation: 1 memory word = ______16________(in bytes)
d. Mention the names of major registers used by your code generator:
    - For base address of an activation record: __NA______
    - for stack pointer:_________________RSP_______________
    - others (specify):____All of the general purpose registers____________
e. Mention the physical sizes of the integer, real and boolean data as used in your code generation module
size(integer): _2____(in words/locations), _________32______(in bytes)
size(real): ____4_____(words/locations), __________64____(in bytes)
size(booelan): ____1__(in words/ locations), ________16_______(in bytes)

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

g. Specify the following:

- Caller's responsibilities:**NA**
- Callee's responsibilities:**NA**

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

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

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

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

l. register allocation (your manually selected heuristic) : **all local variables and temporaries are stored in memory, if we need to perform some operations on these, only then are the registers populated with the data from these memory locations**

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

n. Where are you placing the temporaries in the activation record of a function? **Towards the top of the activation record, first all the local variables find their places, then the temporaries.**
(note that we have only implemented code generation for the driver function, so the answers above and applicable to driver function only)

11. **Compilation Details**:

a. Makefile works (yes/No):_____Yes_______

b. Code Compiles (Yes/ No):_____Yes__________

c. Mention the .c files that do not compile:__________________NA____________________________

d. Any specific function that does not compile:_______________NA___________________________

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

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) _________7829_____________ and (in seconds) _____0.007829_____________

ii. t2.txt (in ticks) __7882____________________ and (in seconds) _____0.00788________________

iii. t3.txt (in ticks) ______8408________________ and (in seconds) ______0.008408_____________

iv. t4.txt (in ticks) _______8456_______________ and (in seconds) __0.008456_________________

v. t5.txt (in ticks) _______3434_______________ and (in seconds) ____0.003434_______________

vi. t6.txt (in ticks) ______7441________________ and (in seconds) _____0.007441_____________

vii. t7.txt (in ticks) ____10144__________________ and (in seconds) _____0.010144______________

viii. t8.txt (in ticks) ____9981__________________ and (in seconds) ____0.009981_______________

ix. t9.txt (in ticks) ____12102__________________ and (in seconds) ___0.0012102_____________

x. t10.txt (in ticks) ___10794___________________ and (in seconds) __0.0010794______________

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)
The code generation module is not fully implemented, we could only handle assignment statements (handling boolean and arithmetic constructs only), io statements, for loop, and array references.
15. Are you availing the lifeline (Yes/No): ____Yes__________
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 -felf64 code.asm && gcc code.o &&./a.out___________________________________
____
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: ____________No____
____
____
19. Declaration: We, ________________Shaurya Marwah, Ruchir Kumbhare, Ashwin Murali, Hari Sankar, Dilip Venkatesh____ __ ______________________ _________________________________ (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:__2019B3A70459P______________Name:_____Shaurya_Marwah___________________________________

ID:__2019B5A70650P______________Name:_____Ruchir_Kumbhare__________________________________

ID:__2019B2A70957P______________Name:_____Ashwin_Murali_____________________________________

ID:__2019B3A70564P______________Name:_____Hari_Sankar_______________________________________

ID:__2020A7PS1203P______________Name:_____Dilip_Venkatesh__________________________________

Date: ____13/4/2023__________ Group number _________29__________

Should not exceed 6 pages.