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 9

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 : **2020A7PS0092P** Name : **Sarthak Shah**

ID : **2020A7PS1675P** Name : **Bhanupratap Singh Rathore**

ID : **2020A7PS0072P** Name : **Archaj Jain**

ID : **2020A7PS0098P** Name : **Siddharth Khandelwal**

ID : **2020A7PS0108P** Name : **Rishi Rakesh Shrivastava**

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

1 ast.c	9 doublyLinkedList.h	17 intermediateCodeGen.h	25 parser.h
2 ast.h	10 driver.c	18 intermediateCodeGenDef.h	26 parserDef.h
3 astdef.h	11 hashTable.c	19 lexer.c	27 parseTree.c
4 codegen.c	12 hashTable.h	20 lexer.h	28 parseTree.h
5 codegen.h	13 hashTabledef.h	21 lexerDef.h	29 remove_comments.c
6 doublyLinkedList.c	14 intermediateCodeGen.c	22 parser.c	30 removeComments.h
7 setADT.c	15 setADT.h	23 stackADT.c	31 stackADT.h
8 symbol_table_def.h	16 symbol_table.c	24 symbol_table.h	32 makefile
33 astrules.pdf	34 coding_Details(stage 2).pdf	35 DFA.pdf	36 First and Follow.pdf
37 grammar.txt	38-48 c1.txt to c11.txt	49-58 t1.txt to t10.txt	

3. Total number of submitted files: **58** (including all testcases and codingDetails proforma)
4. Have you mentioned names and IDs of all team members at the top of each file (and commented well)? (Yes/ no): **YES**
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 test cases (C#.txt, #:1-11): **No**

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): c11.txt

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   a. AST node structure→ contains a pointer to parent contains child info and sibling info the structure is similar to parse tree structure

   b. Symbol Table structure: -> Contains child symbol table array, mod wrapper, no of childs, pointer to parent symbol table and hashtable of symbols

   c. array type expression structure: contains bool flag is_dynamic and range of the array

   d. Input parameters type structure: it is an Linked List of symbol stored in module symbols

   e. Output parameters type structure:it is an Linked List of symbol stored in module symbols

   f. Structure for maintaining the three address code(if created) :it is an quadruple structure containing operator info arg1 arg2 and result's symbol and node

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 present

   c. Number and type of input and output parameters : traversal of AST of input and output parameter list and the actual parameters in second pass of type checking (only thing done in pass2 is type checking and no of parameters for formal and actual parameters, even declaration checks in use module statement done in pass1)

   d. assignment of value to the output parameter in a function : used a bool flag is_assigned in symbol

   e. function call semantics: handled in second pass of type checking no. of parameters checked, type mismatch checked, not declared checked, output var should not be array checked, also if module declared and defined before call checked.

   f. static type checking : all static arrays type checked based on structural equivalence, bound checking done type mismatch checked

   g. return semantics: return var should not be array checked by is_array, it should be assigned checked by flag is _assigned.

   h. Recursion : checked through comparing mod wrapper of that statement and module called in that statement.

   i. module overloading : module entry already present in global symbol table

   j. 'switch' semantics : switch variable should be a variable of type Integer or Boolean, case variable type should be matched to switch variable type by using type_inh in ast_node, default checked in the end if

boolean type switch variable there should be no default. if any error in switch variable the cases are further not explored and if there is type mismatch b/w case and switch var that particular case is not explored.

k. 'for' and 'while' loop semantics: for loop var checked as type should be integer, range should be integer checked, for loop var should not be assigned checked through is_assigned flag in symbol, while loop vars checked for their declaration, if all variables are not assigned checked through maintaining a list of while loop variable symbols and finally checking there is_assigned

l. handling offsets for nested scopes: offsets are passed through recursion in nested scopes and are continuous in a module.

m. handling offsets for formal parameters:offsets are added in module start and these are considered to be pass by value except in case of array where it is handled as you specified in mail.

n. handling shadowing due to a local variable declaration over input parameters: If symbol table entry found in current scope, use that definition otherwise use definition of input parameter list of module entry from global symbol table.

o. array semantics and type checking of array type variables: array type checked if dynamic or not, two dynamic arrays assignment not checked currently as that may be allowed in runtime but in case of static array assignment range diff checked as specified by you, type of array checked. In array access, in case of static access bound checking done. Array not in return list of function handled.

p. Scope of variables and their visibility :scope is handled through symbol tables, diff symbol tables for each scope. visibility also handled through symbol tables where children tables can access parent symbol table symbols. Scope also maintained through recursion for the line no. The ending scopes were calculated in ast generation and starting in symbol_table generation.

q. computation of nesting depth: passed in recursion, increased by one whenever a new symbol_table is being created.

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 = 1(in bytes)
    d. 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):_____
    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), _____2_____(in bytes)
       size(real): _____4_____(in words/ locations), ____4_____(in bytes)
       size(boolean): _____1_____(in words/ locations), _____1_____(in bytes)

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

    g. Specify the following:
        - Caller's responsibilities:_____
        - Callee's responsibilities:_____

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

       _____

       _____

_____

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

j. How is a dynamic array parameter receiving its ranges from the caller? ___NOT IMPLEMENTED____

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

l. register allocation (your manually selected heuristic) :_____
_____

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

n. Where are you placing the temporaries in the activation record of a function? _____In the Bottom of Symbol Table ___

11. **Compilation Details**:
    a. Makefile works (Yes/No) : **Yes**
    b. Code Compiles (Yes/ No) : **Yes**
    c. Mention the .c files that do not compile : **NIL**
    d. Any specific function that does not compile:____NIL_____
    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) **3687** and (in seconds) **0.003687**
    ii. t2.txt (in ticks) **3620** and (in seconds) **0.003620**
    iii. t3.txt (in ticks) **4802** and (in seconds) **0.004802**
    iv. t4.txt (in ticks) **5498** and (in seconds) **0.005498**
    v. t5.txt (in ticks) **4788** and (in seconds) **0.004788**
    vi. t6.txt (in ticks) **7799** and (in seconds) **0.007799**
    vii. t7.txt (in ticks) **5940** and (in seconds) **0.005940**
    viii. t8.txt (in ticks) **6075** and (in seconds) **0.006075**
    ix. t9.txt (in ticks) **7075** and (in seconds) **0.007075**
    x. t10.txt (in ticks) **4941** and (in seconds) **0.004941**
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) : **CODE GENERATION**
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 -no-pie code.o -o code && ./code

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

17. Any other point you wish to mention: ____The part of code written works perfectly like we never faced issue from parser and lexer while working on semantic analysis so the code is robust but we were not able to finish code gen due to time limitation. ___
18. Declaration: We, **Sarthak Shah** , **Rishi Rakesh Shrivastva** , **Bhanupratap Rathore** , **Archaj Jain** , **Siddharth Khandelwal** 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 : 2020A7PS0092P	Name: **Sarthak Shah**
ID : 2020A7PS0108P	Name: **Rishi Rakesh Shrivastva**
ID : 2020A7PS1675P	Name: **Bhanupratap Rathore**
ID : 2020A7PS0072P	Name: **Archaj Jain**
ID : 2020A7PS0098P	Name: **Siddharth Khandelwal**

Date: 13/04/2023 Group number: **9**

Should not exceed 6 pages.