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 :14

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 n

   ID: 2020A7PS0075P Name: SHREYAS KETKAR

   ID: 2020A7PS0133P Name: MRIDUL CHANDAK

   ID: 2020A7PS0144P Name: VARUN SAHNI

   ID: 2020A7PS0974P Name: KUSHAGRA SAHNI

   ID: 2020A7PS0983P Name: CHIRAG MAHESHWARI

2. Mention the names of the Submitted files (Include Stage-1 and Stage-2 both)
   1 driver.c
   2 lexer.c
   3 lexer.h
   4 lexerDef.h
   5 hastTable.c
   6 hashTable.h
   7 hashTableDef.h
   8 parser.c
   9 parser.h
   10 parserDef.h
   11 tree.c
   12 tree.h
   13 treeDef.h
   14 stack.c
   15 stack.h
   16 stackDef.h
   17 grammarSymbol.h
   18 symbolTable.c
   19 symbolTable.h
   20 symbolTableDef.h
   21 ast.c
   22 ast.h
   23 astDef.h
   24 makefile

3. Total number of submitted files: _24________ (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 2 as per the details uploaded)
   g. Code Generator (Yes/No):____no_____________

7. **Execution Status**:
   a. Code generator produces code.asm (Yes/ No):___________No______________
   b. code.asm produces correct output using NASM for testcases (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):____

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)

a. AST node structure: it contains information regrading the label which tell what kind of information it stores, it also has a reference to its children , siblings, it also stores a boolean which tell if it is a leaf node and also the information about the token if it is a leaf.

b. Symbol Table structure:Contains two tables one is the function symbol table (the first symbol table) in which each entry is a function symbol table entry which is hashed and contains a pointer to the symbol table. Each symbol table contains pointers to symbol table entries and the symbol table entries contain the required information.

c. array type expression structure: Used the existing data structures to implement the given functionality

d. Input parameters type structure: The function symbol contains a parameter Symbol table entry which stores all the required information for the input list

e. Output parameters type structure: Same structure as the input parameter list.

f. Structure for maintaining the three address code(if created) : NA

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 : Checked from the hash table used in the symbol table and function symbol table

b. Multiple declarations: Checked from the hash table used in the symbol table and function symbol table

c. Number and type of input and output parameters:Match the types of actual and formal parameters

d. assignment of value to the output parameter in a function Match the types

e. function call semantics: Check the parameter types using function symbol table

f. static type checking : Ensured by comparing the types of the variables using the entries in the symbol table

g. return semantics: has been handled by use of calling conventions which specify how return values are returned to the calling function.

h. Recursion : check for existence in functional symbol tables.

i. module overloading:check in function symbol table

j. 'switch' semantics :created new nested symbol table for each case and handled default case as well

k. 'for' and 'while' loop semantics: created a new nested symbol table

l. handling offsets for scopes: handled in the declare stmt by adding offsets to the required tokens

m. handling offsets for formal parameters:handled in the declare stmt by adding offsets to the required tokens

n. handling shadowing due to a local variable declaration over input parameters:handled by overwriting the input parameters

o. array semantics and type checking of array type variables: ___________________________________

____

p. Scope of variables and their visibility : Implemented by the nesting level parameter stored in each symbol table.

q. computation of nesting depth: The depth of each symbol table is computed by adding 1 to the depth of its parent symbol table while traversal

10. Code Generation: NA

(Module Not Implemented)

a. NASM version as specified earlier used (Yes/no):_______________________

b. Used 32-bit or 64-bit representation:____

c. For your implementation: 1 memory word = ____________________(in bytes)

d. Mention the names of major registers used by your code generator:

- For base address of an activation record: __________________
- for stack pointer:______________________________________
- others (specify):_______________________________________

e. Mention the physical sizes of the integer, real and boolean data as used in your code generation module
size(integer): _______________________(in words/ locations), _______________(in bytes)
size(real): __________________________(in words/ locations), _______________(in bytes)
size(booelan): _______________________(in words/ locations), _______________(in bytes)

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

____

____

____

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? ____

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

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

n. Where are you placing the temporaries in the activation record of a function? __________________
____

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) ______________________ and (in seconds) _____________________

    ii. t2.txt (in ticks) ______________________ and (in seconds) _____________________

    iii. t3.txt (in ticks) ______________________ and (in seconds) _____________________

    iv. t4.txt (in ticks) ______________________ and (in seconds) _____________________

    v. t5.txt (in ticks) ______________________ and (in seconds) _____________________

    vi. t6.txt (in ticks) ______________________ and (in seconds) _____________________

    vii. t7.txt (in ticks) ______________________ and (in seconds) _____________________

    viii. t8.txt (in ticks) ______________________ and (in seconds) _____________________

    ix. t9.txt (in ticks) ______________________ and (in seconds) _____________________

    x. t10.txt (in ticks) ______________________ and (in seconds) _____________________

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)
    ____
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]
    ____NA____
    ____
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: To run the code, the following information is also present on the top of the driver function. To compile - simply type make. To execute, type - './a.out test_case_name.txt outputfile.txt'. The outputfile would simple be a file in which certain extra outputs would be handled (to handle correctness).
19. Declaration: We, Varun Sahni, Mridul Chandak, Shreyas Ketkar, Kushagra Sahni, Chirag Maheswari (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: 2020A7PS0075P Name: SHREYAS KETKAR
ID: 2020A7PS0133P Name: MRIDUL CHANDAK
ID: 2020A7PS0144P Name: VARUN SAHNI
ID: 2020A7PS0974P Name: KUSHAGRA SAHNI
ID: 2020A7PS0983P Name: CHIRAG MAHESHWARI

Date: 12th April 2023 Group number 14

Should not exceed 6 pages.