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: 1

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: 2020A7PS1205P	Name: Nikhil Pradhan
ID: 2020A7PS0146P	Name: Toshit Jain
ID: 2020A7PS0116P	Name: Ansh Gupta
ID: 2020A7PS0035P	Name: Shreekar Puranik
ID: 2020A7PS1209P	Name: Sriram Ramanathan

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

1. lexerDef.h	7. parser.h	13. ast.c	19. grammar.txt
2. parserDef.h	8. ast.h	14. typeChecker.c	20. DFA_scan.pdf
3. astDef.h	9. typeChecker.h	15. driver.c	21. firstfollow.pdf
4. symbolTableDef.h	10. hash.c	16. makefile	22. astRules.pdf
5. hash.h	11. lexer.c	17. README.txt	23. t1 - t16.txt
6. lexer.h	12. parser.c	18. Proforma.pdf	

3. Total number of submitted files: **38**
4. Have you mentioned names and IDs of all team members at the top of each file (and commented well)? **Yes**
5. Have you compressed the folder as specified in the submission guidelines? **Yes**

6. **Status of Code development**: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
    a. Lexer: **Yes**
    b. Parser: **Yes**
    c. Abstract Syntax tree: **Yes**
    d. Symbol Table: **Yes**
    e. Type checking Module: **Yes**
    f. Semantic Analysis Module: **Yes** (reached LEVEL **4** as per the details uploaded)
    g. Code Generator: **No**

7. **Execution Status**:
    a. Code generator produces code.asm: **No**
    b. code.asm produces correct output using NASM for testcases: **NA**
    c. Semantic Analyzer produces semantic errors appropriately: **Yes**
    d. Static Type Checker reports type mismatch errors appropriately: **Yes**

e. Dynamic type checking works for arrays and reports errors on executing code.asm: **No**

f. Symbol Table is constructed: **Yes** and printed appropriately: **Yes**

g. AST is constructed: **Yes** and printed: **Yes**

h. Name the test cases out of 21 as uploaded on the course website for which you get the segmentation fault: **NA**

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
    a. AST node structure: It includes Label (Node type), token, type information (stores primitive type, whether it is primitive, static array or dynamic array, array bounds), child pointer, sibling pointer, scope and nesting level
    b. Symbol Table structure: 2 types of tables – main table and module table
        i. Main table structure: module name, input parameter list, output parameter list, pointer to module table
        ii. Module table structure: variable name, nesting level, scope, width, offset, type information (same as ASTNode), is_changed flag, type of variable(input, output, for loop or normal)
    c. array type expression structure: (array name, bounds, primitive type)
    d. Input parameters type structure: parameter name and type info
    e. Output parameters type structure: parameter name and type info
    f. Structure for maintaining the three address code: 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 : Symbol table entry empty
    b. Multiple declarations: Symbol table already populated with matching scope entry
    c. Number and type of input and output parameters: traversal of linked list of parameters and respective types from main symbol table
    d. assignment of value to the output parameter in a function: keeping a flag for modification of output parameters
    e. function call semantics: Searches module name in main symbol table. Checks number and type input and output parameters.
    f. static type checking: checks type in the symbol table
    g. return semantics: populate flags for ensuring no output parameters are shadowed
    h. Recursion: checks if module's name in function call matches the current function
    i. module overloading: use symbol table to check no re-declaration of same module name
    j. 'switch' semantics: checked types and presence of default
    k. 'for' and 'while' loop semantics: used the is_changed flag to check the loop semantics
    l. handling offsets for nested scopes: assigned offsets in order of declaration
    m. handling offsets for formal parameters: offsets are first assigned to input and output parameters
    n. handling shadowing due to a local variable declaration over input parameters: search function returns variable with highest nesting level that satisfies the scope condition. Input parameters have lowest nesting level hence they get shadowed.

o. array semantics and type checking of array type variables: checked bounds of static arrays

p. Scope of variables and their visibility : populated scope during construction of AST using line number of def, for, while, case, end.

q. computation of nesting depth: computed during construction of AST by adding 1 whenever we encountered a loop, module, or switch case

10. Code Generation:
    a. NASM version as specified earlier used: **NA**
    b. Used 32-bit or 64-bit representation: **NA**
    c. For your implementation: 1 memory word = **NA**
    d. Mention the names of major registers used by your code generator:
        - For base address of an activation record: **NA**
        - for stack pointer: **NA**
        - others (specify: **NA**
    e. Mention the physical sizes of the integer, real and boolean data as used in your code generation module
       size(integer): **1** (in words/ locations), **NA** (in bytes)
       size(real): **2** (in words/ locations), **NA** (in bytes)
       size(boolean): **4** (in words/ locations), **NA** (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?: **both local variables and parameters**

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

    m. Which primitive data types have you handled in your code generation module?: **NA**
    n. Where are you placing the temporaries in the activation record of a function?: **NA**

11. **Compilation Details**:
    a. Makefile works: **YES**

    b. Code Compiles: **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]: **NA**

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): **30** and (in seconds): **0.03**

    ii. T2.txt (in ticks): **29** and (in seconds): **0.029**

iii. T3.txt (in ticks): **28** and (in seconds): **0.028**

iv. T4.txt (in ticks): **70** and (in seconds): **0.07**

v. T5.txt (in ticks): **50** and (in seconds): **0.05**

vi. T6.txt (in ticks): **69** and (in seconds): **0.069**

vii. T7.txt (in ticks): **65** and (in seconds): **0.065**

viii. T8.txt (in ticks): **72** and (in seconds): **0.072**

ix. T9.txt (in ticks): **116** and (in seconds): **0.116**

x. T10.txt (in ticks): **38** and (in seconds): **0.038**

13. **Driver Details**: Does it take care of the **TEN** options specified earlier?: **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: **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: Test cases t1 to t10 are for semantic analysis and test cases t11 to t16 are for syntactical analysis
19. Declaration: We, Nikhil Pradhan, Toshit Jain, Ansh Gupta, Shreekar Puranik, Sriram Ramanathan 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: 2020A7PS1205P Name: Nikhil Pradhan
ID: 2020A7PS0146P Name: Toshit Jain
ID: 2020A7PS0116P Name: Ansh Gupta
ID: 2020A7PS0035P Name: Shreekar Puranik
ID: 2020A7PS1205P Name: Sriram Ramanathan (Group leader)

Date: 12-04-2023 Group number: 1

Should not exceed 6 pages.