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

| Group Number |
| --- |
| 28 |

**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: 2016B4A70614P | Name: Anirudh Buvanesh |
| ID: 2016B4A70824P | Name: Anubhav Bansal |
| ID: 2016B5A70716P | Name: Aayush Agarwal |
| ID: 2016B5A70607P | Name: Manan Agarwal |
| ID: 2016B4A70636P | Name: Ashish Kumar |

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

| | | |
| --- | --- | --- |
| 1 lexer.c | 7. SymbolTable.c | 13 codegen.c |
| 2 lexerDef.h | 8. SymbolTableDef.h | 14 codegen.h |
| 3 lexer.h | 9. SymbolTable.h | 15 driver.c |
| 4 AST.c | 10 parser.c | 16 makefile |
| 5 ASTDef.h | 11 parser.h | 17 grammer.txt |
| 6 AST.h | 12 parserDef.h | |

3. Total number of submitted files: 17 (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] 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 testcases (C#.txt, #:1-11): For all test cases(1-11)
    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): yes
    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): N/A

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   a. AST node structure:  Contains all the necessary information about Token value of the Parse tree node
   b. Symbol Table structure: There are two types of symbol tables. One for function entries (func_st) and one for variables(var_st). A single function symbol table entry contains many var_st. Each symbol table has line number, hash table and depth related to it's scope.
   c. array type expression structure: Consists of primitive datatype, range variables or values and an is_static flag.
   d. Input parameters type structure:  Linklist with attributes- Datatype, Lexeme, Line Number, next pointer
   e. Output parameters type structure: Linklist with attributes- Datatype, Lexeme, Line Number, next pointer
   f. Structure for maintaining the three address code(if created) :linked list
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 and cannot be shadowed
   c. Number and type of input and output parameters: Match with the linklist of corresponding parameters
   d. assignment of value to the output parameter in a function: Keeping a count of number output parameters assigned and then matching with the number of output parameters
   e. function call semantics: Check that function is defined/declared before calling. Also check that the invoking variables are defined and match with input parameters
   f. static type checking: Traversing the Ast in post-order and assigning a type to every subexpression and then comparing the types of required expressions
   g. return semantics: Traversing the list of output parameter list and comparing it with the list of actual parameters and checking if the actual parameter variables are declared
   h. Recursion: Check that the called module's lexeme does not match with the ongoing module
   i. module overloading: Entry for module already exists in the Symbol Table
   j. 'switch' semantics: Creating a single scope for all cases. Checking for allowed datatypes (i.e. Integer and boolean) of switch variable. Checking that whether the type of case values match with the type of switch identifier. Taking care of when a default should be present and when not.
   k. 'for' and 'while' loop semantics: For variable should be an integer and it should neither be declared not assigned inside the for scope. While expression should be of boolean type, at least one identifier in the while expression must be assigned at least once.
   l. handling offsets for nested scopes: Maintained through a global offset value
   m. handling offsets for formal parameters: Add offset of 5 for arrays (both static and dynamic) and add offset equal to width for primitive data types
   n. handling shadowing due to a local variable declaration over input parameters: Check that the entry in the symbol table for the variable has the flag equal to 1, which denotes that It is an input parameter
   o. array semantics and type checking of array type variables:  Primitive datatype comparison for dynamic arrays is done at compile time else all type checks for dynamic arrays is done at runtime.  For static arrays primitive datatype comparison and bounds check for index, both are done at compile time.

Checking whether identifier used as an index is integer is done at compile time for both dynamic and static arrays.

p. Scope of variables and their visibility: Implemented using a hierarchical symbol table structure, which compares the line number of declaration and use for choosing the correct declaration.

q. computation of nesting depth: Depth of a symbol table is filled while its creation by adding one to the nesting depth of parent. Depth of root variable symbol table is initialized to 1.

10. Code Generation:
    a. NASM version as specified earlier used (Yes/no): yes
    b. Used 32-bit or 64-bit representation: 32-bit
    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: ebp
        - for stack pointer: esp
        - 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): two bytes | (in words/ locations), 2 | (in bytes) |
| size(real): four bytes | (in words/ locations), 4 | (in bytes) |
| size(booelan): one byte | (in words/ locations), 1 | (in bytes) |

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

        Formal parameters are copied to space of local parameters and when the function finished then it is the responsibility of the called function to copy back the values and return value back to the calling function

    g. Specify the following:
        - Caller's responsibilities: copy the formal parameters to space of local parameters
        - Callee's responsibilities copy the values of formal parameters and return values back to calling function

    h. How did you maintain return addresses? (write 3-5 lines): maintaining the value in the register.

    i. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? Yes, in the registers

    j. How is a dynamic array parameter receiving its ranges from the caller? By copying the values

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

        Yes

    l. register allocation (your manually selected heuristic) : Specific registers are used for specific purposes with eax and ebx for storing left and right operands of any operator.

    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 registers

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

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) | 1552 | and (in seconds) | 0.001552 |
| ii. | t2.txt (in ticks) | 1399 | and (in seconds) | 0.001399 |
| iii. | t3.txt (in ticks) | 2591 | and (in seconds) | 0.002591 |
| iv. | t4.txt (in ticks) | 2271 | and (in seconds) | 0.002271 |
| v. | t5.txt (in ticks) | 2711 | and (in seconds) | 0.002711 |
| vi. | t6.txt (in ticks) | 3500 | and (in seconds) | 0.003500 |
| vii. | t7.txt (in ticks) | 3918 | and (in seconds) | 0.003918 |
| viii. | t8.txt (in ticks) | 4745 | and (in seconds) | 0.004745 |
| ix. | t9.txt (in ticks) | 6287 | and (in seconds) | 0.006287 |
| x. | t10.txt (in ticks) | 1786 | and (in seconds) | 0.001786 |

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)
    N/A
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 elf32 code.asm –o code.o
    gcc –m32 code.o –o code.out
    ./code.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 **(All are true)**
18. Any other point you wish to mention:

    We have implemented expression evaluation on REAL numbers as well.

19. Declaration: We, (your names) Anirudh Buvanesh, Anubhav Bansal, Aayush Agarwal, Manan Agarwal, Ashish Kumar 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: 2016B4A70614P | Name: Anirudh Buvanesh |
| ID: 2016B4A70824P | Name: Anubhav Bansal |
| ID: 2016B5A70716P | Name: Aayush Agarwal |
| ID: 2016B5A70607P | Name: Manan Agarwal |

ID: 2016B4A70636P Name: Ashish Kumar

Date:20/04/2020

---

Should not exceed 6 pages.