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

Group Number
15

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: 2017A7PS0164P NAME: DISHITA MALAV

   ID: 2017A7PS0040P NAME: SHAILY BHATT

   ID: 2017A7PS0139P NAME: SHEFALI TRIPATHI

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

1. lexer.c	7. ast.c	13. semanticAnalyser.c	19. MakeFile
2. lexer.h	8. ast.h	14. semanticAnalyser.h	20. driver.c
3. lexerDef.h	9. astDef.h	15. semanticAnalyserDef.h	21. c1.txt to c11.txt
4. parser.c	10. symbolTable.c	16. codeGen.c	22. t1.txt to t10.txt
5. parser.h	11. symbolTable.h	17. codeGen.h	23. Coding Details
6. parserDef.h	12. symbolTableDef.h	18. inputGrammar.txt	

3. Total number of submitted files: 42 (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, c5.txt, c6.txt and c8.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): c7.txt, c10.txt and c11.txt

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   a. AST node structure: __Contains a tag_label (of enum) and pointer to parent, first_child and next_sibling in the tree. Also contains a pointer to token which points to appropriate tokens in case of teminals and is NULL in other cases____
   b. Symbol Table structure: Hash table implemented in the form of an array of Linked Lists, of a HashEntry which stores a tagged union of structs namely IDNode, FuncNode and StmtNode.
   c. array type expression structure: The struct for all variable identifiers is the same, called IDNode. If the variable identifier is an array struct parameter isArray = true and all the other parameters related to the Array are populated otherwise not.
   d. Input parameters type structure: The struct for all variable identifiers is the same, called IDNode. Input parameters list is implemented as a Linked List of IDNodes.
   e. Output parameters type structure: The struct for all variable identifiers is the same, called IDNode. Output parameters list is implemented as a Linked List of IDNodes.
   f. Structure for maintaining the three address code(if created) : None

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 : Retrieval from symbol table in that scope and recursively in its parent scopes till the global scope returns a NULL value
   b. Multiple declarations: While populating symbol table, if the entry already exists then multiple declaration error is thrown
   c. Number and type of input and output parameters: Stored in a linked list, nodes of the linked list contain all the type information etc., pertaining to the respective parameters.
   d. assignment of value to the output parameter in a function: During the traversal of a module, whenever value is assigned to an output parameter, isAssigned flag is set to true. When the traversal reaches the end of a function, the list of output parameters is traversed and isAssigned flag is checked. If it is false, error is reported
   e. function call semantics: Existence of functions is checked from global symbol tale and number and type of formal and actual parameters is compared
   f. static type checking : The index value is compared with the lower and upperbound of static array as retrieved from symbol table
   g. return semantics: number and type of formal and actual parameters is compared
   h. Recursion: Checks the name of the caller module, if it matches with the called module, an error is thrown.
   i. module overloading: Checks the symbol table entry, which contains a variable isDefined and isDeclared, which is set to true when a module is defined and/or declared respectively. If either of them is set to true, corresponding error is thrown.
   j. 'switch' semantics : type of case variables should be same as that of switching variable. In case of integer type, default statement should be present and in case of boolean it should be absent
   k. 'for' and 'while' loop semantics: Iterator variable of for is int, expression for while should be boolean
   l. handling offsets for nested scopes: Offsets of a variable in a function is calculated from the beginning of the definition of the function, not including the offset due to the input/output parameters.

m. handling offsets for formal parameters: Offsets are calculated starting from 0 and incremented whenever a new formal parameter is encountered.

n. handling shadowing due to a local variable declaration over input parameters: Input parameters are not hashed into the local symbol table of a module unlike output parameter, hence input parameters can be overshadowed by local variables.

o. array semantics and type checking of array type variables: All information is maintained in a struct called IDNode.

p. Scope of variables and their visibility : Variables declared in a nested scope are visible within that scope. Variables of parent scope are visible in their nested scopes. Variables declared within a nested scope can overshadow variables in parent scope.

q. computation of nesting depth: Done as per specifications, stored a parameter in IDNode.

10. Code Generation:
    a. NASM version as specified earlier used (Yes/no): Yes
    b. Used 32-bit or 64-bit representation: 32-bit representation
    c. For your implementation: 1 memory word = 2 (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):_____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), 2 (in bytes)
size(real): 2 (in words/ locations), 2 (in bytes)
size(booelan): 1 (in words/ locations), 2 (in bytes)

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

We didn't implement.

    g. Specify the following:

- Caller's responsibilities: N/A
- Callee's responsibilities: N/A

    h. How did you maintain return addresses? (write 3-5 lines): N/A
    i. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? N/A
    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): Only local variables.
    l. register allocation (your manually selected heuristic): Used Available registers
    m. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): Integer, Boolean and Real (Only handled declaration of real).
    n. Where are you placing the temporaries in the activation record of a function? N/A

11. **Compilation Details**:
    a. Makefile works (yes/No): Yes
    b. Code Compiles (Yes/ No): Yes
    c. Mention the .c files that do not compile: All compile
    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) _____2571_____ and (in seconds) ____0.002571_____
    ii. t2.txt (in ticks) ____2437_____ and (in seconds) ____0.002437_____
    iii. t3.txt (in ticks) ____7324_____ and (in seconds) ____0.007324_____
    iv. t4.txt (in ticks) ____5487_____ and (in seconds) ____0.005487_____
    v. t5.txt (in ticks) ____8453_____ and (in seconds) ____0.008453_____
    vi. t6.txt (in ticks) ____8071_____ and (in seconds) ____0.008071_____
    vii. t7.txt (in ticks) ____8282_____ and (in seconds) ____0.008282_____
    viii. t8.txt (in ticks) ____11793_____ and (in seconds) ___0.011793_____
    ix. t9.txt (in ticks) ____12971_____ and (in seconds) ____0.012971_____
    x. t10.txt (in ticks) ___4751_____ and (in seconds) ____0.004751_____

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 for Arrays with dynamic bounds and calling of module is not done.
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]:
    (Assuming the output file generated is code.asm)
    nasm -f elf -o code.o code.asm
    ld -m elf_i386 -s -o code code.o
    ./code

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: We have been unable to print strings on console.

    True is printed as 1 and false as 0.

    No "input" "output" strings are printed.

    Only scanning and printing happens for the get_value or print statements, respectively, corresponding to the code.

    In case of a run time error, -1 is printed and code exits

19. Declaration: We, Dishita Malav, Shaily Bhatt and Shefali Tripathi (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: 2017A7PS0164P Name: DISHITA MALAV
    ID: 2017A7PS0040P Name: SHAILY BHATT
    ID: 2017A7PS0139P Name: SHEFALI TRIPATHI

Date: 21 April 2020

Should not exceed 6 pages.