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

Group Number

2

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:2017A7PS0001P	Name: Bhoomi Sawant
ID:2017A7PS0086P	Name: Pratik Kakade
ID:2017A7PS0114P	Name: Yashdeep Gupta
ID:2017A7PS0116P	Name: Ayush Singhal
ID:2017A7PS0218P	Name: Saksham Gupta

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

1. astGenerator.c	9. grammar.txt	17. makefile	25. stack.c
2. astGenerator.h	10. grammar_Init.c	18. parser.c	26. stack.h
3. ASTNodeDef.h	11. grammar_Init.h	19. parser.h	27. stackDef.h
4. codeGenerator.c	12.grammar_InitDef.h	20. parserDef.h	28. symbolTable.c
5. codeGenerator.h	13. lexer.c	21. semanticAnalyzer.c	29. symbolTable.h
6. driver2.c	14. lexerDef.h	22. semanticAnalyzer.h	30. symbolTableDef.h
7. typeChecker.c	15. typeChecker.h	23 . t1-t10.txt	31.Coding_Details.pdf
8. lexer.h	16. c1-c11.txt	24. Sample_Symbol_table.txt	

3. Total number of submitted files: 50
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/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) : **YES**
   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) : **NONE**

8. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   a. AST node structure: Main structure consists of union of all the different types of nodes, a tag to identify type of node, an integer to store line number and a pointer to the variable entry in case it is a variable
   b. Symbol Table structure: It is implemented as a hash array(SymbolTable), hashed on the function name, with each node pointing to a function table which contains entries for a given function along with a pointer to local tables for its scope.
   c. array type expression structure: Array type consists of an enum to identify its type, arrayFlag to indicate if its an array, union of string and integer to store the lower and upper bound along with the flags to indicate which to use, pointers to lower and upper bound variable entries and flag to check if its static or dynamic
   d. Input parameters type structure: It is implemented as a linked list with structure to store lexeme, type, line number, etc.
   e. Output parameters type structure:It is implemented as a linked list with structure to store lexeme, type, line number, etc.
   f. Structure for maintaining the three address code(if created) : Not created

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 for variable entry in symbol table
   b. Multiple declarations: Symbol table already found populated
   c. Number and type of input and output parameters : Traversed linked list of parameters and check type
   d. assignment of value to the output parameter in a function : Pop and move value to given offset
   e. function call semantics: Push parameters in reverse, call function, pop parameters
   f. static type checking : Bound and type checking for static arrays
   g. return semantics: Check if output parameters are all assigned value
   h. Recursion : Compared function call name with current function name
   i. module overloading: Checked if module definition already exist
   j. 'switch' semantics : No real values, Default present if integer, No default in case of boolean
   k. 'for' and 'while' loop semantics: For loop variable not changed inside loop, While loop variable changes
   l. handling offsets for nested scopes: Offset starts from previous declared variable in outer scope
   m. handling offsets for formal parameters: Input parameters start from 0, followed by output parameters
   n. handling shadowing due to a local variable declaration over input parameters: First output parameters are checked, then local variables then input parameters for symbol table entry
   o. array semantics and type checking of array type variables: Lower bound<= Higher, array = array allowed if same types and bounds, bound checking for access using index

p. Scope of variables and their visibility : Scope and visibility from start of block to end, overshadowed by local parameters

q. computation of nesting depth : traverse Local table tree and storing depth of tree with root depth 1

10. Code Generation:

a. NASM version as specified earlier used (Yes/no): Yes
b. Used 32-bit or 64-bit representation: 64-bit
c. For your implementation: 1 memory word = 8 (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): RAX,RBX,RCX,RDX,RSI,RDI,R8,R10,XMM0,XMM1
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), 16 (in bytes)
size(real): 4 (in words/ locations), 32 (in bytes)
size(boolean): 1 (in words/ locations), 8 (in bytes)

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

Function calls were implemented by first pushing output parameters from right to left onto the stack, followed by pushing input parameters from right to left, followed by a call to the function which pushes the return address onto the stack , then pushing RBP to the stack at the start of the function. Thus, Local variables are stored at RBP-8-offset and parameters are stored at RBP+16+offset. Before returning, RBP is popped, followed by a function call to "ret". After that, the caller must pop all parameters in the reverse order of pushing.

g. Specify the following:

   - Caller's responsibilities: Pushes output parameters right to left, followed by input parameters from right to left, and then calls the callee (pushing the return address onto the stack).

   - Callee's responsibilities : Pushes RBP onto the stack at the start of the function, and pops it before returning.

h. How did you maintain return addresses? (write 3-5 lines): We have used the inbuilt function calls namely "call function" and "return" for maintaining return addresses. Upon "call function", the return address is pushed onto the stack. And, it is used at the end by calling ret.

i. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? Parameter passing is done by first pushing output parameters, followed by input parameters, so inside the callee, the parameters can be accessed using the memory location [RBP+16+offset].

j. How is a dynamic array parameter receiving its ranges from the caller? For array parameters, first high is pushed, followed by low, followed by the array base address. The callee accesses the low and high from the formal parameters as [RBP+16+arrayOffset+8] and [RBP+16+arrayOffset+8+16].

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

l. register allocation (your manually selected heuristic: RDX was used to maintain the size of dynamic memory allocated in each scope and to free it at the end of the scope. It has been preserved throughout each scope. For arithmetic calculations, temporaries as well as registers like RAX, RBX, RCX, XMM0 etc. have been used as and when needed. Temporaries have been recycled across various functions. RBP stores the base of the activation record of the callee and RSP stores the top of the stack.

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

n. Where are you placing the temporaries in the activation record of a function? The maximum number of temporaries are calculated before code generation and are declared as fixed data memory which can be used by any 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: **None**
    d. Any specific function that does not compile: **None**
    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):2703	and (in seconds) : 0.002703
ii.	t2.txt (in ticks):2478	and (in seconds): 0.002478
iii.	t3.txt (in ticks): 3690	and (in seconds): 0.003690
iv.	t4.txt (in ticks): 3136	and (in seconds): 0.003136
v.	t5.txt (in ticks): 3684	and (in seconds): 0.003684
vi.	t6.txt (in ticks): 6060	and (in seconds): 0.006060
vii.	t7.txt (in ticks): 5199	and (in seconds): 0.005199
viii.	t8.txt (in ticks): 6005	and (in seconds): 0.006005
ix.	t9.txt (in ticks): 7518	and (in seconds): 0.007518
x.	t10.txt (in ticks): 2686	and (in seconds): 0.002686

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) : **None**
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 -felf64 out.asm -o out.o && gcc -no-pie out.o && ./a.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
18. Any other point you wish to mention : Our code also implements float type operations. It also avoids output of error in case of reuse of undeclared variables in a given function similar to gcc. Further, during run time, we also give errors in case of out of bound error instead of a simple segmentation fault.
19. Declaration: We, Bhoomi Sawant, Pratik Kakade, Yashdeep Gupta, Ayush Singhal, Saksham Gupta 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:2017A7PS0001P Name:Bhoomi Sawant

ID:2017A7PS0086P	Name:Pratik Kakade
ID:2017A7PS0114P	Name:Yashdeep Gupta
ID:2017A7PS0116P	Name:Ayush Singhal
ID:2017A7PS0218P	Name:Saksham Gupta

Date: 20/04/2020

Should not exceed 6 pages.