**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 : 24**

*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: 2020A7PS0024P Name: Nachiket Kotalwar**

**ID: 2020A7PS0026P Name: Parth Patel**

**ID: 2020A7PS0043P Name: Partha Sarthi Purkayastha**

**ID: 2020A7PS0045P Name: Labeeb Ahsan**

**ID: 2020A7PS0975P Name: Anish Atul Kulkarni**

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

| 1. ast.c | 1. ast.h | 1. ast.txt | 1. astDef.h |
| --- | --- | --- | --- |
| 1. astrules.txt | 6-15. c1txt-c10.txt | 16. chash.c | 16. chash.c |
| 17. chash.h | 18. codegen.c | 19. codegen.h | 20. codgenDef.h |
| 21. coding\_details\_(stage 1)-Group24.docx | 22. coding\_details\_(stage 1)-Group24.docx | 23. driver.c | 24. enum.h |
| 25. generateQuad.c | 26. grammar.txt | 27. intermediateCodegen.c | 28. intermediateCodegen.h |
| 29. intermediateCodegenDef.h | 30. lexer.c | 31. lexer.h | 32. lexerDef.h |
| 33. makefile | 34.parseTree.c | 35. parseTree.h | 36. parser.c |
| 37. parser.h | 38. parserDef.h | 39. semanticAnalyzer.c | 40.semanticAnalyzer.h |
| 41. semanticAnalyzerDef.h | 42. symbolTable.c | 43. symbolTable.h | 44.DFA\_Group24.pdf |
| 45. ASTRules\_Group24.pdf | 46. GrammarRules\_Group24.pdf | 47.- 56. t1.txt - t10.txt |  |

1. Total number of submitted files: 56 (All files should be in **ONE** folder named exactly as Group number)
2. 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]
3. Have you compressed the folder as specified in the submission guidelines? (yes/no) **Yes**
4. **Status of Code development**: Mention 'Yes' if you have developed the code for the given module, else mention 'No'.
   1. Lexer (Yes/No): **Yes**
   2. Parser (Yes/No): **Yes**
   3. Abstract Syntax tree (Yes/No): **Yes**
   4. Symbol Table (Yes/ No): **Yes**
   5. Type checking Module (Yes/No): **Yes**
   6. Semantic Analysis Module (Yes/ no): **Yes** (reached LEVEL 4 as per the details uploaded)
   7. Code Generator (Yes/No): **Yes**
5. **Execution Status**:
   1. Code generator produces code.asm (Yes/ No): **Yes**
   2. code.asm produces correct output using NASM for testcases (C#.txt, #:1-11): : c1.txt - c3.txt
   3. Semantic Analyzer produces semantic errors appropriately (Yes/No): **Yes**
   4. Static Type Checker reports type mismatch errors appropriately (Yes/ No): **Yes**
   5. Dynamic type checking works for arrays and reports errors on executing code.asm (yes/no): **No**
   6. Symbol Table is constructed (yes/no) **Yes** and printed appropriately (Yes /No): **Yes**
   7. AST is constructed (yes/ no) **Yes** and printed (yes/no) **Yes**
   8. 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
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure : It contains node label, child nodes, pointer to symbol table if a scope starts there, lexeme (if applicable) and start and end bounds of scope (if applicable).
   2. Symbol Table structure : Contains pointer to hash\_table containing symbol entries and pointer to parent symbol table. Contains nesting level, offset, certain flags (like forloop, whileloop), start line and end line.
   3. array type expression structure: Contains flag to check if array is static/dynamic. If bounds are static numerical values along with sign is stored.
   4. Input parameters type structure: Contains an array of types. Each type further contains flags and values associated with the parameter.
   5. Output parameters type structure: Contains an array of types. Each type further contains flags and values associated with the parameter.
   6. Structure for maintaining the three address code(if created) : Quadruple having arg1, arg2, operator, and pointer to symbol table entry of result.
7. **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.]
   1. Variable not Declared : Symbol table entry not found
   2. Multiple declarations: Symbol table entry already found.
   3. Number and type of input and output parameters: traversal of array of parameters and respective types
   4. assignment of value to the output parameter in a function : Checking the assigned value of a symbol table entries of output IDs on exiting a module.
   5. function call semantics:Declaration and type checking of variables in idlist on the left side, if any. Type checking of actual and formal parameters ,checking number of parameters in the call checking,number of variables to be assigned.
   6. static type checking : Array bound checks if static, expression type checks during assign statements etc
   7. return semantics:Declaration and type checking of variables in idlist on the left side, if any.
   8. Recursion : Not allowing module to call itself if the scope module name is the same as current module.
   9. module overloading: Checking if name is same but parameters are different.
   10. 'switch' semantics : Type check of switch variable. If switch variable is boolean , children of case\_stmt are checked to ensure that no default statement exists. Similar check to enforce default is done in case of integers.
   11. 'for' and 'while' loop semantics: 1. for loop- create a new scope and declare index as integer and ensure that its not modified inside. 2. While- expression is ensured to be of type boolean and new scope is created locally.
   12. handling offsets for nested scopes: Offsets of modules start from 0. In case of other scopes, like for,switch case etc, continue the offset of the module.
   13. handling offsets for formal parameters :Offsets for them are added to the function’s activation record initially. Then, declarations inside the function are considered if any.
   14. handling shadowing due to a local variable declaration over input parameters: We allow the input parameters to be shadowed but output parameters cannot be re-declared inside the same module scope(cannot be shadowed).
   15. array semantics and type checking of array type variables: First the expression inside the array is ensured to be an integer. If the array is static, bound checking is done. If the id is not an array giving an index to it is not allowed. Arrays while assigning are checked to be equivalent in width and type.
   16. Scope of variables and their visibility : Static scoping is implemented. Every module , individual switch case, for loop, while loop have their own scopes. Variable is first searched in local scope and then the parent scope. The same local scope cannot have two variables of the same id.
   17. computation of nesting depth: nesting depth is initially zero. When a new scope is created it is equal to the nesting level of the parent +1.
8. Code Generation:
   1. NASM version as specified earlier used (Yes/no): **Yes**
   2. Used 32-bit or 64-bit representation: **64-bit** representation
   3. For your implementation: 1 memory word = 8 (in bytes)
   4. 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, RDX
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): 1 (in words/ locations), 8 (in bytes)

size(real): 1 (in words/ locations), 8 (in bytes)

size(booelan): 1 (in words/ locations), 8 (in bytes)

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

-

* 1. Specify the following:
     + Caller's responsibilities: -
     + Callee's responsibilities:-
  2. How did you maintain return addresses? (write 3-5 lines): -
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? -
  4. How is a dynamic array parameter receiving its ranges from the caller? -
  5. What have you included in the activation record size computation? (local variables, parameters, both): Both
  6. register allocation (your manually selected heuristic) : -
  7. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): Integer, real and boolean
  8. Where are you placing the temporaries in the activation record of a function? At the top of the stack

1. **Compilation Details**:
   1. Makefile works (yes/No): **Yes**
   2. Code Compiles (Yes/ No): **Yes**
   3. Mention the .c files that do not compile: - None
   4. Any specific function that does not compile: - None
   5. Ensured the compatibility of your code with the specified versions [GCC, UBUNTU, NASM] (yes/no) **Yes**
2. Execution time for compiling the test cases [lexical, syntax and semantic analyses including symbol table creation, type checking and code generation] :
   * 1. t1.txt (in ticks)\_\_\_\_\_\_\_\_\_\_\_\_\_\_28310\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_0.02831\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     2. t2.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_21659\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_0.021659\_\_\_\_\_\_\_\_\_\_\_\_
     3. t3.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_32422\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_0.032422\_\_\_\_\_\_\_\_\_\_\_\_
     4. t4.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_29083\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_0.029083\_\_\_\_\_\_\_\_\_\_\_\_
     5. t5.txt (in ticks) \_\_\_\_\_\_\_\_\_43227\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_0.043227\_\_\_\_\_\_\_\_\_\_\_
     6. t6.txt (in ticks) \_\_\_\_\_\_\_\_\_40236\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_0.040236\_\_\_\_\_\_\_\_\_\_\_
     7. t7.txt (in ticks) \_\_\_\_\_\_\_\_49364\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_0.049364\_\_\_\_\_\_\_\_\_\_
     8. t8.txt (in ticks) \_\_\_\_\_\_\_\_19576\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_0.019576\_\_\_\_\_\_\_\_\_\_\_\_
     9. t9.txt (in ticks) \_\_\_\_\_\_\_\_\_28554\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_0.028554\_\_\_\_\_\_\_\_\_\_\_
     10. t10.txt (in ticks) \_\_\_\_\_\_\_\_\_10193\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_0.010193\_\_\_\_\_\_\_\_\_\_
3. **Driver Details**: Does it take care of the **TEN** options specified earlier?(yes/no): **Yes**
4. Specify the language features your compiler is not able to handle (in maximum one line):

Segmentation fault in dynamic bound checking.

1. Are you availing the lifeline (Yes/No): **Yes**
2. 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 elf64 -o code.o code.asm

gcc -no-pie -o code code.o

./code

1. **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 ,
2. Any other point you wish to mention: None
3. Declaration: We, **Nachiket Kotalwar, Parth Patel , Partha Sarathi Purkayastha, Labeeb Ahsan and Anish Kulkarni**, 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: 2020A7PS0024P Name: Nachiket Kotalwar

ID: 2020A7PS0026P Name: Parth Patel

ID: 2020A7PS0043P Name: Partha Sarthi Purkayastha

ID: 2020A7PS0045P Name: Labeeb Ahsan

ID: 2020A7PS0975P Name: Anish Atul Kulkarni

Date: 12 April 2023 Group number : 24

---------------------------------------------------------------------------------------------------------------------------------------------

Should not exceed 6 pages.