# 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: 23 (Write your group number here)**

*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: 2020A7PS0073P | Name: Shashank Agrawal |
| ID: 2020A7PS0096P | Name: Akshat Gupta |
| ID: 2020A7PS0129P | Name: Anish Ghule |
| ID: 2020A7PS0134P | Name: Shadan Hussain |
| ID: 2020A7PS1513P | Name: Tarak P V S |

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

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | driver.c | 11 | parser.c | 21 | testcase5.txt | 31 | FirstFollowSets.pdf |
| 2 | global.h | 12 | parser.h | 22 | testcase6.txt | 32 | ast_semantic_rules.pdf |
| 3 | grammar.txt | 13 | parseTree.c | 23 | proforma.docx | | Also some testcases |
| 4 | hashmap.c | 14 | parseTree.h | 24 | ast.h | | |
| 5 | hashmap.h | 15 | removeComments.c | 25 | ast.c | | |
| 6 | lexer.c | 16 | removeComments.h | 26 | symbolTable.c | | |
| 7 | lexer.h | 17 | testcase1.txt | 27 | symbolTable.h | | |
| 8 | linkedList.c | 18 | testcase2.txt | 28 | codegen.h | | |
| 9 | linkedList.h | 19 | testcase3.txt | 29 | codegen.c | | |
| 10 | makefile | 20 | testcase4.txt | 30 | dfa.pdf | | |

4. Total number of submitted files: 59 (including testcases not mentioned above)(All files should be in **ONE** folder named exactly as Group number)
5. Have you mentioned names and IDs of all team members at the top of each file (and commented well)? (Yes/ no) Yes(in all code files) [Note: Files without names will not be evaluated]
6. Have you compressed the folder as specified in the submission guidelines? (yes/no)______Yes_________

7. **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, implemented in symbol table module
   f. Semantic Analysis Module (Yes/ no):Yes (reached LEVEL 4 as per the details uploaded)
   g. Code Generator (Yes/No):Yes

8. **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
   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):No segmentation faults in compilation
9. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   a. AST node structure: AST node stores the node information( token, Variable number) and pointers to its parent sibling and first child.
   b. Symbol Table structure: Symbol Table is a HashMap DS, which stores hash of lexeme names along with other info (offset, width, data types, line no, scope, pointer to another symbol table if needed) related to that lexeme. Lexeme may be a construct(for/while/case..etc) or a module name or a variable name.
   c. array type expression structure: consists of scope of module, scope of array variable, and type structure of the array(which includes bounds, lexemes of bounds in case of dynamic arrays, and type of array element.
   d. Input parameters type structure: stored all input parameters in input_plist which is list of pointers to all astNodes of input parameters. The pointer to this list is stored in the astNode of module.
   e. Output parameters type structure: similar to that of the input parameter list
   f. Structure for maintaining the three address code(if created) : Not created.
10. **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: Entry not found in current scope and parent scopes

b. Multiple declarations: Entry already present with same name in current scope

c. Number and type of input and output parameters: maintaining a datatype list corresponding to datatypes of input and output list in symbol table entry for module.

d. assignment of value to the output parameter in a function maintained a flag to check if assigned with each variable

e. function call semantics: Checked the types of the actual parameters with the input_plist symbol Table entry

f. static type checking: For assignment, the type of the variable (from symbol table) on the left side of assignment is checked with the datatype that is generated on evaluating the expression on the right side. For each operation, the left and right operands are checked for validity and the type generated is passed in a bottom up manner recursively.

g. return semantics: checked the types of returned values using output_plist from symbol table , each returned variable is cheked for assignment using a Boolean flag ifAssigned

h. Recursion: Checked the parent module in which a call to a module is made, if the IDs of both match, report error.

i. module overloading: Checked for duplicate ID in the global symbol table that contains an entry for each module

j. 'switch' semantics : The id on which switch is declared should be either Boolean or integer (using Symbol table). The type of each case should match with the type of the id on which switch is declared. Default case must be present if id is integer else must not be present.

k. 'for' and 'while' loop semantics: For- LoopVar flag set true for loop variable . Every assignment in the scope of for is checked to ensure that loop variable is not assigned. While- Expression type checked to be boolean. If any id is present in the expression, at least one of them should be assigned in the loop.

l. handling offsets for nested scopes: offset for each variable is maintained in the symbol table entry and table maintains the offset for the stack top.

m. handling offsets for formal parameters: For each module, the local offset starts with zero and the symbol table keeps track of the next possible offset.

n. handling shadowing due to a local variable declaration over input parameters: NOT Handled, as initially we assumed that input parameters and local variables share the same scope, and designed our structures accordingly, which could not be changed later on.

o. array semantics and type checking of array type variables: Type checking is done for both static and dynamic arrays(rules in assignments). Bound checking is done for static arrays at compile time and dynamic arrays at run time.

p. Scope of variables and their visibility : The variables declared in the current scope are visible in any nested scope unless shadowed and invisible in any parent scope.

q. computation of nesting depth: Nesting depth of every module is initialized to 1, and as nested scopes are added the nesting depth of nested scope is 1 + nesting depth of current scope.

11. Code Generation:
    a. NASM version as specified earlier used (Yes/no): Yes
    b. Used 32-bit or 64-bit representation:64
    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, rbs for temporaries and rcx as a counter if required_

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), ____________8___(in bytes)
   size(real): ____________1______________(in words/ locations), __________8_____(in bytes)
   size(booelan): __________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 are implemented using call and ret. Whenever there is use module, we call the label generated for that module. The stack moves to that point and after its compilating, stack pointer move back to original space.

g. Specify the following:

   - Caller's responsibilities:Pushes its base pointer return address and input parameters. Allocates space for output parameters.
   - Callee's responsibilities: Updates output parameters

h. How did you maintain return addresses? (write 3-5 lines):

   The return address is pushed on to the stack when we use call.

i. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? They are passed onto the stack along with the base address.

j. How is a dynamic array parameter receiving its ranges from the caller? It receives base address lower and upper bounds.

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

l. register allocation (your manually selected heuristic) : We allocate space on the stack by using RSP register.

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

n. Where are you placing the temporaries in the activation record of a function?
   Temporaries are pushed on the stack when required.

12. **Compilation Details**:
    a. Makefile works (yes/No): yes
    b. Code Compiles (Yes/ No): yes
    c. Mention the .c files that do not compile: no such file
    d. Any specific function that does not compile: everything compiles
    e. Ensured the compatibility of your code with the specified versions [GCC, UBUNTU, NASM] (yes/no) yes

13. 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) 3019 and (in seconds) .003019
    ii. t2.txt (in ticks) 4587 and (in seconds) .004587
    iii. t3.txt (in ticks) 7849 and (in seconds) .007849
    iv. t4.txt (in ticks) 5434 and (in seconds) .005434
    v. t5.txt (in ticks) 8846 and (in seconds) .008846
    vi. t6.txt (in ticks) 5867 and (in seconds) .005867

vii. t7.txt (in ticks) 7857 and (in seconds) .007857

viii. t8.txt (in ticks) 8890 and (in seconds) .008890

ix. t9.txt (in ticks) 9858 and (in seconds) .009858

x. t10.txt (in ticks) 13101 and (in seconds) .013101

14. **Driver Details**: Does it take care of the **TEN** options specified earlier?(yes/no):_Yes__
15. Specify the language features your compiler is not able to handle (in maximum one line)
    Floating point arithmetic could not be handled due to different register file.
16. Are you availing the lifeline (Yes/No): No, (Used in first submission)
17. 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 code.asm && gcc -no-pie code.o -o code && ./code**
18. **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 + Almost nothing hardcoded, well commented
19. Any other point you wish to mention: NA
20. Declaration: We, Shashank Agrawal, PVS Tarak, Shadan Hussain, Akshat Gupta, Anish Ghule(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 | 2020A7PS0073P | Name: Shashank Agrawal |
| ID | 2020A7PS0096P | Name: Akshat Gupta |
| ID | 2020A7PS0129P | Name: Anish Ghule |
| ID | 2020A7PS0134P | Name: Shadan Hussain |
| ID | 2020A7PS1513P | Name: Tarak P V S |

Date: 12-04-23 Group number 23

---

Should not exceed 6 pages.