**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

Group Number

**8**

**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: 2017A7PS0093P Name: Ayush Jain

ID: 2017A7PS0025P Name: Bharat Bhargava ID: 2017A7PS0117P Name: Satvik Golechha

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

1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 13\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 19\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 8\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 14\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 9\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 15\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 16\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 22\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 17\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 23\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 12\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 18\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 24\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Total number of submitted files: \_\_\_\_\_\_\_\_\_\_\_ (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): Within symbol table module itself
   6. Semantic Analysis Module (Yes/ no): Yes (reached LEVEL \_\_\_\_ 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): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   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):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   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):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure: It removes unnecessary nodes (i.e., nodes which won’t be used in future) from the generated parse tree and produces a syntax tree occupying less memory and suitable enough to perform semantic analysis and code generation subsequently.
   2. Symbol Table structure:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. array type expression structure:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. Input parameters type structure:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  3. Output parameters type structure: Linked list of
  4. Structure for maintaining the three address code(if created) : We performed code generation directly; did not go through intermediate code generation

1. **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 :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   2. Multiple declarations: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. Number and type of input and output parameters:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. assignment of value to the output parameter in a function \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   5. function call semantics:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   6. static type checking :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   7. return semantics:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   8. Recursion :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   9. module overloading:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   10. 'switch' semantics :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   11. 'for' and 'while' loop semantics: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   12. handling offsets for nested scopes:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   13. handling offsets for formal parameters:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   14. handling shadowing due to a local variable declaration over input parameters:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   15. array semantics and type checking of array type variables: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Scope of variables and their visibility :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. computation of nesting depth: No check (supports nesting depth till exhaustion of memory)

1. Code Generation:
   1. NASM version as specified earlier used (Yes/no): Yes
   2. Used 32-bit or 64-bit representation: 64-bit
   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): rdi, rsi, rdx, rcx, r8-r15 for function call and return
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): ½ (in words/ locations), 4 (in bytes)

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

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

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

As soon as the function is called, the instruction pointer (for return address) is automatically pushed onto the stack. We then push rbp, make space for input and output parameters, push rbp again two times and then make space for local variables (for we know the number and types of local variables in a function, which are stored in the corresponding symbol table). On return, we first destroy the local variables and then the parameters. Finally, just before return, we pop the original base pointer into rbp.

* 1. Specify the following:
     + Caller's responsibilities: Pass on the input parameters for callee in registers.
     + Callee's responsibilities: Keep space for input and output parameters and modify memory locations keeping track of the computations performed with the parameters/ local variables.
  2. How did you maintain return addresses? (write 3-5 lines): As mentioned in Q.10 f., we push the value of rbp on the stack, just after we call a function. We ensure that the statements executed subsequently do not modify this memory location. Upon completion of the execution of the function, we pop into rbp the previous base pointer (the original base value is thus preserved) and return to the next instruction to be executed.
  3. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee?: We first ensured that no statement other than “module reuse statement” required any dedicated register. Then, when we call a function, we put all the input parameters to be passed on from the caller in registers (thus implicitly getting an upper bound on the number of input parameters which can be passed). In the callee’s activation record, we’ve reserved the space for parameter allocation, where we directly put (or, rather, save the states) the values of input parameters and retrieve from the values of the output parameters into registers. We then ensure that these registers finally put the values in appropriate memory locations in caller’s activation record.
  4. How is a dynamic array parameter receiving its ranges from the caller? For each variable present in the range construct in the AST, we have allotted space in the stack. So, we’re directly receiving the value of that (range) variable from the stack itself.
  5. What have you included in the activation record size computation? (local variables, parameters, both):

Local variables only (for parameters, we’re simply reserving constant space in the stack)

* 1. register allocation (your manually selected heuristic) : No heuristic employed. For statements other than module reuse statement, it doesn’t matter at all what registers we’re using, as per our implementation. For the module reuse statement, though, we have maintained dedicated registers for saving values of input parameters, which are rdi, rsi, rdx, rcx, r8, r9, r10, r11, r12, r13, r14, r15 in order.
  2. Which primitive data types have you handled in your code generation module?(Integer, real and boolean): Integer and Boolean. The instructions for handling real constants and variables are found to have varied outputs in our implementation (may be, due to inadequate support of 8087 on our devices, for 80x86 family cannot handle FP arithmetic on its own). At the same time, the constraints of the stack and the size of the registers for FP arithmetic require unnecessary pushes and pops and breaking a single constant into two, hence we cannot perform an operation directly in memory (especially the qword/ dword operations).
  3. Where are you placing the temporaries in the activation record of a function?: We’ve used registers for temporaries. We haven’t placed them in the activation record.

1. **Compilation Details**:
   1. Makefile works (yes/No):\_\_\_\_\_\_\_\_\_\_\_
   2. Code Compiles (Yes/ No):\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. Mention the .c files that do not compile:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. Any specific function that does not compile:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   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) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     2. t2.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     3. t3.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     4. t4.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     5. t5.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     6. t6.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     7. t7.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     8. t8.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     9. t9.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
     10. t10.txt (in ticks) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and (in seconds) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. **Driver Details**: Does it take care of the **TEN** options specified earlier?(yes/no):\_\_\_\_\_\_\_\_\_\_\_
4. Specify the language features your compiler is not able to handle (in maximum one line)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Are you availing the lifeline (Yes/No): No (already availed)
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 code.asm && gcc code.o –o code && ./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: N/A
3. Declaration: We, Ayush Jain, Bharat Bhargava and Satvik Golechha (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: 2017A7PS0093P Name: Ayush Jain

ID: 2017A7PS0025P Name: Bharat Bhargava ID: 2017A7PS0117P Name: Satvik Golechha

Date: 20/4/2020

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

Should not exceed 6 pages.