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

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

Group Number

34

**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:2017A7PS0237P Name : KRISHNA SURVARAPU

ID:2017A7PS0076P Name: SHIVA KRISHNA

ID: 2017A7PS0037P Name: AQEEL SHAIK

ID:2017A7PS0222P Name: ABHISHEK REDDY

ID:2017A7PS0119P Name:SURYA SUMANTH MEESALA

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

1 ast.c 7 lexer.c 13 parserDef.h 19 stack.c

2 ast.h 8 lexer.h 14 parsetree.h 20 stack.h

3 astdef.h 9 lexerDef.h 15 parsetreedef.h 21 stackdef.h

4 driver.c 10 makefile 16 semantics.c [ 22-31] t1.txt – t10.txt

5 intermediatecode.c 11 parser.c 17 semantics.h 32. grammar.txt

6 intermediatecode.h 12 parser.h 18 semanticsdef.h 33. coding\_details[stage\_2].docx

1. Total number of submitted files: 32 (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 \_\_\_\_ as per the details uploaded)
   7. Code Generator (Yes/No):NO(only partially done)
5. **Execution Status**:
   1. Code generator produces code.asm (Yes/ No):NO
   2. code.asm produces correct output using NASM for testcases (C#.txt, #:1-11):NO
   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): t1,3 gives straight segmentation fault
6. **Data Structures** (Describe in maximum 2 lines and avoid giving C definition of it)
   1. AST node structure contains label,lexval,childhead,sibling and symboltableentrypointer
   2. Symbol Table structure: a hash table that basically holds a linked list of symbol arrays which are nothing but a linked list
   3. array type expression structure: [type , startindex,endindex,dynamic,index]
   4. Input parameters type structure: handled using symboltable entries by keeping track of their number
   5. Output parameters type structure: handled using symboltable entries by keeping track of their number
   6. Structure for maintaining the three address code(if created) : A string which is given to every ast node maintains intermediate code
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 : no entry in symbol table
   2. Multiple declarations: if already present in symbol table then
   3. Number and type of input and output parameters: the parameters are stored in the functions’ symbol table
   4. assignment of value to the output parameter in a function: the type is being checked
   5. function call semantics: input and output parameters specified should exist and should match the type
   6. static type checking : symbol table entry has a variable named type in it
   7. return semantics: output parameters specified should exist and should match the type
   8. Recursion : module scope of the function in symbol table should not be same as the function used
   9. module overloading: cannot declare the same module twice
   10. 'switch' semantics : value should be of type INTEGER and not an array
   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 : implemented using a linked list of symbol table
  2. computation of nesting depth: depth of parent+1

1. Code Generation:
   1. NASM version as specified earlier used (Yes/no):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   2. Used 32-bit or 64-bit representation:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. For your implementation: 1 memory word = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in bytes)
   4. Mention the names of major registers used by your code generator:

* For base address of an activation record: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* for stack pointer:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* others (specify):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  1. Mention the physical sizes of the integer, real and boolean data as used in your code generation module

size(integer): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in bytes)

size(real): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in bytes)

size(booelan): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(in words/ locations), \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(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): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

* 1. How have you maintained parameter passing? How were the statically computed offsets of the parameters used by the callee? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  2. How is a dynamic array parameter receiving its ranges from the caller? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  3. What have you included in the activation record size computation? (local variables, parameters, both): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  4. register allocation (your manually selected heuristic) :\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  5. Which primitive data types have you handled in your code generation module?(Integer, real and boolean):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
  6. Where are you placing the temporaries in the activation record of a function? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Compilation Details**:
   1. Makefile works (yes/No): YES
   2. Code Compiles (Yes/ No):YES
   3. Mention the .c files that do not compile: --
   4. Any specific function that does not compile: GETONETOKEN() not working correctly with multiline comments
   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): Yes\_
4. Specify the language features your compiler is not able to handle (in maximum one line)

Multiline Comments

1. Are you availing the lifeline (Yes/No):NO
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]

./compiler ti.txt {where i is the testcase to be used}

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: We tried our level best to implement but not able to generate final code but we are able to generate intermediate code which we tried to tweak to convert it to machine code but it is not complete .
3. Declaration: We, Krishna Suravarapu , Shiva Krishna , Abhishek Reddy , Aqeel Shaik, Surya Sumanth Meesala (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 2017A7PS0237P Name: KRISHNA SURAVARAPU

ID 2017A7PS0076P Name: SHIVA KRISHNA

ID 2017A7PS0037P Name: AQEEL SHAIK

ID 2017A7PS0119P Name: SURYA SUMANTH MEESALA

ID 2017A7PS0222P Name: ABHISHEK REDDY

Date: 20/04/2020

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

Should not exceed 6 pages.