**CSCI 465 Fall 2020**

Exam 2: **TAKE-HOME**

Total Points: 120

Total number of questions: 7

**Due date: December 15 at 12:00 p.m.**

Name:\_\_\_Derek Trom\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Note: Your answers must be typed and spellchecked. To answer the questions, you are allowed to consult textbook, lecture notes, your own note, published articles in Journals/Conferences. However, this is individual work meaning that you are NOT allowed to share your solution with other students whatsoever. Also, you are encouraged to use proper citations/references to refer the resources that you have read/used to formulate your solutions.*

1. ( 10 points)You need to briefly answer the following questions regarding your the compiler project:
   1. What was the main obstacle when you first started working on your project?
      1. The main obstacle when I started working on this was picking the correct language and which tools I wanted to use for the project.
   2. Which delivery was the easiest one and how long did it take you to do it?
      1. The easiest delivery was the lexer which only took me a little over a day off and on to do.
   3. Which delivery was the hardest part and how long did it take you to do it?
      1. The intermediate code generator was the hardest part for me and this took several days.
   4. Of which part of your project is the most rewarding part?
      1. The most rewarding part was seeing the created MIPS code and watching it run correctly in Mars.
   5. Which part of your compiler would you like to re-implement if you had the chance to improve *your compiler*?
      1. I would reimplement the parser using some sort of tool to make it less time consuming than writing by hand.
   6. What are the lessons you learn from your compiler? What would you do differently if you are asked to construct yet another compiler next time?
      1. Spend more time researching tools that are out there to do some of the work in the background it would save a lot of time and lines of code.
   7. What type of discussion would like to be added/deleted from the course so the course becomes more interesting and meaningful?
      1. I actually enjoyed the discussion in this course. But what I would like to see is maybe 1 or 2 tools put to use in a compiler, because that was the hardest part to figure out.
   8. How big is your compiler in terms of line of code (LOC)?
      1. It was about 4000 lines including comments and blank lines.
   9. If you are given enough time, what would be the additional features you would like to add to your compiler?
      1. I would like to optimize the parser or make it into something other than LL(1) recursive descent.
   10. What is the most important property of a compiler? How did you validate your compiler according to this property?
       1. The most important property was consistency which was validated through many tests and debug statements.
2. (15 points) For each of the following grammar indicate whether overall, general attribute value flow is bottom-up, top-down, left-to-right, and right-to-left.

**(a).**

G→A↓l

A↓n→B↓3n A↓7n

→”c” C↓n-1

B↓n→”a” B↓n+4 “b” C↓2n

→”b”

→”c”

**Top-Down**

**(b)**

G→A↑x

A↑n→B↑u ↑v A↑y [x=uy+v]

→”c” C↑z [x=2z]

B↑v→”a” B↑r↑s“b” C↑x [u=2r+x-s; v=s+1]

→”b” [u=1; v=2]

C↑x →”c” [x=3]

**Bottom-up**

**(c)**

G →A↓ 0 ↑r

A↓x↑z→B↓y ↑z A↓ x↑y

→”c” C↓x↑y [z=10y+3]

B x w → “a” B↓10y+2↑z “b” C↓x↑y [w=10z+1]

→”b” [w=10x+2]

C x y →”c” [y=10x+3]

**This is left-right**

3. (20 points) consider the following grammar:

Terminals = {a, b,c}

Non-Terminals = {S, A}

**Rules:**

(1) S → AA

(2) S→ bc

(3) A→ baA

(4) A→ c

Suppose also we have the following LR(0) states and transitions for the above grammar:



1. Create the LR(0) parse table

Action GOTO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | $ | S | A |
| State 1 |  | S4 | S5 |  | 2 | 3 |
| State 2 |  |  |  | accept |  |  |
| State 3 |  | S7 | S5 |  |  | 6 |
| State 4 | S8 |  | S9 |  |  |  |
| State 5 | R4 | R4 | R4 | R4 |  |  |
| State 6 | R1 | R1 | R1 | R1 |  |  |
| State 7 | S8 |  |  |  |  |  |
| State 8 |  | S7 | S5 |  |  | 10 |
| State 9 | R2 | R2 | R2 | R2 |  |  |
| State 10 | R3 | R3 | R3 | R3 |  |  |

1. Is the above grammar LR(0)? (explain briefly)
   1. Yes I believe this grammar is LR(0) because there are not any Shift Reduce conflicts in the diagram above.
2. (10 points) The following declarations are given for a language that uses name equivalence:

A, D: array [1..100] of int;

C: array [1..100] of int;

F: array [1..100] of int;

Explain briefly: 1) which of the above four variables have the same type, and 2) which ones have different types?

1. Using name equivalence none of these have the same type
2. A,D,C,F are all of different types. If let’s say A was declared and then C was declared as A then they would be equal but none of these are declared as equal to each other.
3. (15 points) Intermediate Representations (IRs) plays a significant role in translating diffident languages working on different platforms.
   1. What are the main properties of Intermediate Representations (IRs)?
      1. Ease of generation
      2. Ease of Manipulation
      3. Procedure Size
      4. Freedom of Expression
      5. Level of abstraction
      6. Found on https://people.cs.umass.edu/~moss/610-slides/17.pdf
   2. Moreover, standardizing IRs has been suggested to address two constant issues in computing industry: 1) software compatibility, and 2) compiler interoperability. Briefly explain how the standardized IRs will address those issues.
      1. Software compatibility can be addressed by basing IR of software on an abstract machine. If this were to be accomplished, code should be able to be run on any machine and thus creating some push in innovation in hardware.
      2. Compiler interoperability can be addressed by also creating a standard IR which would be able to take the strengths of many different compilers and work them into the algorithm of the abstract machine as well. This will also allow for further abstraction of the machine by the way of modularization of the compiler into smaller pieces and more optimization can occure in that way.
      3. Answers above were gathered from the article mentioned below.

**Note**: the complete discussion of the proposed standardization of IR can be found in an article published by Communication of ACM, December 2013, Vol.6, N0.12. The article is available on the blackboard system for the course and can be found under “resources”.

6. (20 points) strongly typed, statically checked languages can help the programmer produce valid programs by detecting large classes of erroneous programs.

a) In what way this feature can improve the compiler’s ability to generate efficient code for a program?

This feature can improve compiler’s ability to generate efficient code because in a strongly typed language the variable is known at compile time which leads to efficiency in code saving the time at runtime from having to decide what it is. Strongly typing also helps because variables are bound to specific data types which can also improve efficiency because the type is known at compile time. This is also can help enable other optimizations within the compiler.

b) Some programming languages either omit declarations or treat them as optional Information. Examples include Scheme program that lacks declarations for variables. Therefore, in the absence of declarations, what element of type system can be used to determine a type for each variable, and in what way it makes harder to implement the language (briefly explain).

I think what Scheme does is considered dynamically typed. Languages like scheme, Python, JavaScript all do this. The way that the compiler does this is to check the types based on the run-time data of the program. This makes it harder to implement the language because the compiler will not know what each type is at compile time, which means everything has to be decided at runtime which can lead to errors being harder to catch and define.

7. ( 30 points) Use the following translation scheme (SDT) and the type **float** [3][4] (i.e., two-dimensional array of type **float**) to perform the following tasks:

1. Generate the parse tree
2. Annotate the tree with attributes (i.e., type and width)
3. Show how the width of the type **float** [3][4] is computed

**Productions Syntax Directed Translation**

**-----------------------------------------------------------**

T🡪 B {t=B.type; w=B.width;}

C {T.type = C .type; T.width = C .width;}

B🡪 **int** {B.type = integer; B.width = 4;}

B🡪 **float** {B.type = float; B.width = 8;}

C🡪 ε {C.type = t; C.width = w;}

C🡪 [**num**] C1 {C.type = array(**num**.value, C1.type); C.width=**num**.value × C1.width; }

1. Width = array(3, array(4, float))
2. Float is 8 width so we replace float with 8.
3. C2.width = Array(3,array(4, 8))
4. C1.width = num.value \* c2.width = 4 \* 8 = 32
5. C.width = num.value \* c1.width = 3 \* 32 = 96
6. T.t = Float, T.w = 96