**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:\_Sam Dressler \_\_\_

*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 you’re the compiler project:
   1. What was the main obstacle when you first started working on your project?

The main obstacle when starting my compiler was figuring out how to make a modular program that would be able to accomplish all the steps in a compiler. The design needed for this project is one that allowed output from the Lex to be input into the intermediate code generator. The output from that step would also need to be sent to the final code generator step. This sequence of programs was difficult for me to implement without writing down a design.

* 1. Which delivery was the easiest one and how long did it take you to do it?

The first delivery was the easiest since it was something that I had some experience doing in a previous course (CSCI 365). I was able to sit down on a Sunday afternoon and complete the first delivery by the early hours of Monday morning.

* 1. Which delivery was the hardest part and how long did it take you to do it?

The second and third deliveries were significantly harder but the most difficult for me was the second. During the development on this delivery, I needed to design many different methods to handle the symbols that were generated by the first delivery. I also needed to generate the three-address code which would be used to generate the final code. The final thing that made this delivery difficult, was developing a method to store variables and ensure that they were in the correct scope. This delivery was only made more difficult by not having a good design going in which resulted in many hours spent debugging and tweaking the outputs of the functions. I spent closer to 25 hours working on the second delivery.

* 1. Of which part of your project is the most rewarding part?

For me, the most rewarding part of the project was developing an actual compiler. Developing a compiler is something that is very difficult to do and was previous overlooked by myself, even though it is something I would use daily. Within the compiler, I am very proud of how I designed the symbol table that is used to store variables once they are declared.

* 1. Which part of your compiler would you like to re-implement if you had the chance to improve *your compiler*?

I would re-implement the intermediate code generation portion of my code if I had the chance. This is because the code is very messy and uses a lot of repeated code that could be refined if given more time. This section of the code also does not utilize the algorithms that were discussed in the course that would have made implementation more scalable. I would also change how the symbol table is structured since implementing while loops proved to be extremely difficult with how my program was structured.

* 1. 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?

Going into such a large project without writing out a detailed design has been a huge mistake. It has led to many hours spent scrutinizing lines of code where there was a bug. If I were to go back and create another compiler or revise the first, I would begin working on it much earlier and the development process would include writing out a very detailed design.

* 1. What type of discussion would like to be added/deleted from the course, so the course becomes more interesting and meaningful?

I think discussions on code samples would greatly benefit the student’s abilities to implement an efficient compiler. However, I understand that each student should use their own creativity to solve the problem. I also think that advice for implementing “make or break” features, such as the symbol table, should be discussed so that when students get to the final delivery they have everything they need to complete the compiler.

Otherwise, I was very happy with the current content of the course.

* 1. How big is your compiler in terms of line of code (LOC)?

The length of my compiler is greater than 2800 lines of code including the driver program, lex, intermediate code generator, and the final code generator.

* 1. If you are given enough time, what would be the additional features you would like to add to your compiler?

If given more time, I would likely implement more robust error messages. Currently some errors will cause the program to stop executing while others will display the required error message and continue executing.

* 1. What is the most important property of a compiler? How did you validate your compiler according to this property?

The most important property of a compiler is generating the correct output. In my compiler, I validated this by having multiple output files that were used to validate the input and output to each of the modules.

1. (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.

Up-arrow = s-attribute, down-arrow = L-attribute

**(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”

This language has an overall general attribute value flow that is *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]

This language has an overall general attribute flow that is *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 language has an overall general attribute flow that is *left-to-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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| State | Action | | | | Goto | |
| ---- | a | b | c | $ | S | A |
| S1 | --- | S4 | S5 | --- | S2 | S3 |
| S2 | --- | --- | --- | Accept | --- | --- |
| S3 | --- | S7 | S5 | --- | --- | S6 |
| S4 | S8 | --- | S9 | --- | --- | --- |
| S5 | R4 | R4 | R4 | --- | --- | --- |
| S6 | R1 | R1 | R1 | --- | --- | --- |
| S7 | S8 | --- | --- | --- | --- | --- |
| S8 | --- | --- | S5 | --- | --- | S10 |
| S9 | R2 | R2 | R2 | --- | --- | ---- |
| S­10 | R3 | R3 | R3 | --- | --- | --- |

1. Is the above grammar LR(0)? (explain briefly)

The above grammar is LR (0) since a LR (0) parser can be created using the shift/reduce method and there are no shift/reduce conflicts. This means that the grammar has no epsilon-productions and is hence LR (0).

1. (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

All the variables have the same type. This type is array [1..100] of int. This is true even between A and D, which are declared in the same line using the same type, and C and F which are declared in following lines but again with the same type. This claim is also backed up by the fact that Pascal is a language that uses non-strict name equivalence. This means that variables declared in separate declarations do not introduce a distinct type.

1. which ones have different types?

None of the arrays have different types since the type of each variable is the same. This means that each of the variables could be assigned to each other. i.e. A := F or D := C.

If we were to define our own array type and recreate the declarations this can be shown more clearly.

I am going to use the following code block to show my thoughts:

Type

my\_array\_type = array [1..100] of int;

Var

A, D : my\_array\_type;

C : my\_array\_type;

F : my\_array\_type;

Each of these variables still have name equivalence as well as semantic equivalence since they are all declared using the same type.

If the variables were to be declared as follows, we would have semantic equivalence, however, we would not have name equivalence since the types are not the same name:

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

C : my\_array\_type;

F : my\_array\_type;

1. \* (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)?

Intermediate Representations (IRs) were created as a method of limiting the complexity of compilers. IR’s can be represented as a data structure with two main properties. The first of these is the IRs ability to provide an accurate translation of a source program without the loss of any information.

The second property its flexibility to handle programs that are written in multiple programming languages. This shows how a higher level of importance is placed on the semantics of a program by breaking the source program into a low-level language that retains the original source program’s purpose. The low-level code that is generated by applying this principle can also change depending on the machine that the compiled program will be running on, further increasing the generality of the IR.

* 1. 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.

The issue of software compatibility is the first main problem that must be overcome in standardizing IRs. The problem arises when two software implementations are non-compatible. This may mean that they have different Instruction Set Architectures (ISAs) or different application binary interfaces (ABIs). This problem can be solved by creating an IR around an abstract machine. This machine will be able to render executable for a particular platform through Ahead-of-time or just-in-time compilation.

The second issue holding back the standardization of Intermediate machines is that of Compiler Interoperability. This problem is defined since there exists no compiler that can handle all the ways in which software and compiler algorithms are written. The solution of this is to adopt multiple standardized compilers that can work together in order to form the output.

**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.

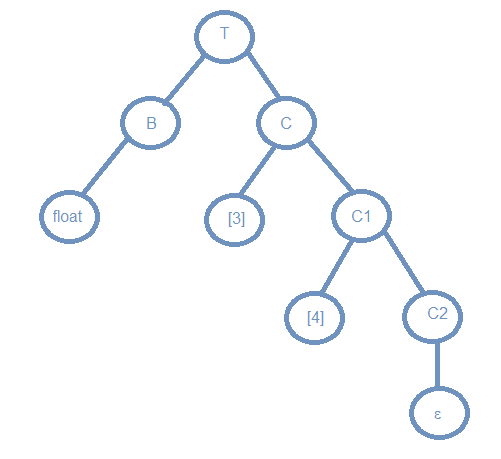
a) In what way this feature can improve the compiler’s ability to generate efficient code for a program?

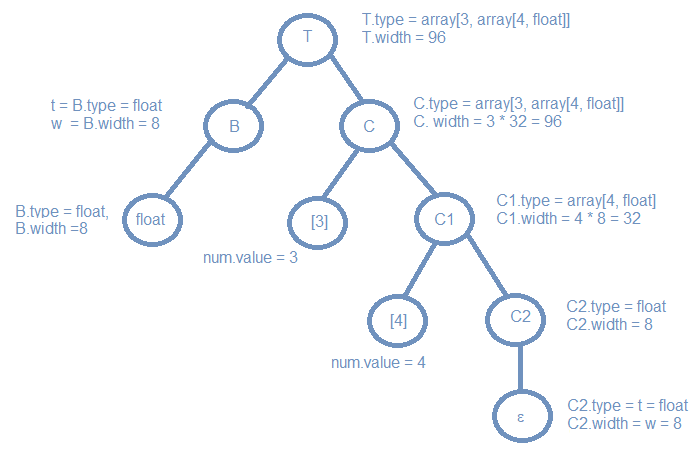
Having a strongly type language increases the efficiency of a compiler’s ability to generate code by checking for errors or exceptions during compile time. Strongly typed language can also be statically or dynamically typed. Being statically typed also provides the advantage of creating a faster running program by allowing for lower-level machine code optimization.

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).

Languages such as scheme, which do not utilize declaration for variables, can be known as dynamically typed languages. In scheme you can declare a variable but must always give it an initial value. When variables are declared, a pointer to an object is what is returned and space for the variable is created. This storage of the variable is known as a binding and is what lets the language know what initial value is stored. Languages that implement this kind of type system are more difficult since they rely on the context that the definition would need to be accounted for. Dynamically typed languages also make it more difficult to find bugs in the source code since there are no generated errors for type changes.

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)



1. Show how the width of the type **float** [3][4] is computed

Width of float [3][4] = num1.value \* (num2.value \* float.width)

= 3 \* (4 \* 8)

= 3 \* (32)

= 96

**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; }