Assignment 3: Three Address Code Generation

Submission Deadline: Tuesday, January 14 (In person at office/drop-box)

Dear students, this assignment description is long and detailed. So read it carefully to understand what to do and what to submit. In addition, there is a quiz bonus component attached to this assignment at the end. Thus, read carefully till the end.

Assignment Task Description

After the midterm exam, we are exploring how syntax directed translations (SDT) rules can be used to collect semantic information about parts of a program and how to generate three address codes for these parts by traversing a syntax tree that we generate during the parsing process. This assignment is about connecting all the dots of SDT rules regarding variable declarations, array element accesses, different types of arithmetic and logic operations, and structuring blocks such as if-else and while loops to generate a complete three address code for a well-known algorithm that you learned in your junior year programming courses. First, let us specify how you should generate the sample source code. Then we will hint how you should connect the different grammar parts given in the reference textbook. Finally, we will discuss how you should address the problem of three address code generation.

Source Program Construction Process:

1. The first two lines of your source code will be as follows. Replace the word ***type*** with float if your student ID is odd. Otherwise, replace it with int.

   type*[100] array;*
   loadArray(array, 10);
2. Then if the last digit of your student ID is larger than 4 then you will write an insertion sort algorithm using while loops that sort the content of the array variable of Step 1. Otherwise, you will write a bubble sort algorithm using while loops.
3. Finally, if the next to last digit of your student ID is odd then the sorting algorithm will sort the array in ascending order. Otherwise, it will sort the array in descending order.

Remember, that you must use while loops – not for loops. There is no discussion of for loops in the reference textbook. Thus, if you have a for loop-based implementation of the sorting algorithm then you cannot generate a three address code for your source.

Process of Connecting Different Grammar Parts of the Textbook:

You start the grammar rules as follows. Here P is the start symbol and represents the entire program and C represents a program component:

P -> C
C -> C C
C -> ε
C -> D
C -> S

Here a component (C) can be variable definition (D) or a statement (S). The first two production rules of C enable you to add as many variable definitions sequence and statement sequences in your source and

mix them. Then you expand D's production rules using the variable definitions grammar given in Section 6.3.3 to 6.3.5 of Chapter 6 of the textbook. For S, you expand the production rules using the statement grammar rules provided in Section 6.4 and Section 6.6.1-6.6.4 of Chapter 6 of the textbook. The production rules of S in the textbook does not allow you to write a function call as a statement. So, add the following set of production rules in the statement (S) and expression (E) related production rules.

S -> E
E -> Func
Func -> id (Params)
Params -> ε
Params -> E, Params
Params -> E

Finally, note that you don't have to write any grammar production rules in your assignment submission. You just follow the instructions to connect the grammars to draw a syntax/parse tree then to traverse it to generate a three-address code.

Three Address Code Generation Process:

To generate the final three address code, you first draw a syntax/parse tree for your source code following the production rules of the grammar (Note that your parse/syntax tree may become too big, and you can draw them in parts to make the drawing fit in the paper.). Then you do a post-order traversal of the parse tree to collect information about the variable definitions in the proper symbol table(s). Then you do another post order traversal from the start symbol (P) to attach Labels and codes to statement blocks using the attribute rules given in Sections 6.6.3 and 6.6.4 of your reference textbook. Notice that attribute rules given in these sections will generate code for statements and attach the code with each statement node of your syntax/parse tree in a code attribute. That is the three address code lines are not actually written on a file when you are doing the post order traversal, rather they are being passed from children to parents. By the way, since the textbook did not describe SDT/SDD rules for function calls, you can generate three-address code for the 'loadArray(array, 10);' source-code line based on your understanding and don't need to explain those few lines. Finally, you just print the code attribute of your start symbol P. That will print the whole three address code for the program.

Submission Script Components

Your submission must have:

1. The original C source code that you are translating. (1 point)
2. The whole syntax/parse tree of the source program. (5 points)
3. The three-address code generated from the tree following the process described above. (4 points)

Note that your submission must be a handwritten version, a typed submission will get a 0 out of 10. Also note that, if you just arbitrarily write a three-address code based on your knowledge of computer architecture course that does not follow the SDD/SDT rules of the textbook then again you will get a zero for the code generation part.

Late Submission Policy: No late submission will be accepted for this assignment as there is a deadline for submitting the final grades of the course within 3/4 days of the final exam of the course.

Quiz Bonus Opportunity

You can earn up to 5 points to compensate for missing points from the quizzes if you can solve the following additional tasks. However, note that I will check each submission manually and will know if you have used AI tools or other means to plagiarize. Then you will get negative marking for each part where you plagiarize. Finally, if you attempt these bonus points then write your solutions for them in one or more separate pages after the completion of the main assignment part.

1. Task 1 (1 point): notice that, I did not add the SDT/SDD rules for code generation for function call grammar part in Page 2. Add proper SDT/SDD rules to that grammar so that you can generate proper code for function calls having multiple parameters following those rules.
2. Task 2 (1 point): add SDT/SDD rules to initialize the symbol table stack, offset stack, and current symbol table variable when three-address code generation starts.
3. Task 3 (1 point): explain how you can modify the grammar (production and SDD/SDT rules) of the book to support variable declarations within loops and conditional branches as supported in regular C++ programs.
4. Task 4 (2 points): Write and explain how you can update the SDD/SDT rules of the reference textbook that described each part separately to have a single whole attribute grammar that will let you do all of type information collection, statement label calculation, and code generation.

Happy Studying Compilation Techniques!