# CSCI 2320: Principles of Programming Language
# Programming Assignment 3: Type System and Semantics

Points: 20 (+ 1 Extra Credit)
Due: Part I (type rules 1 and 2)—Thursday, October 26 (11:59 PM)
The whole assignment—Thursday, November 2 (11:59 PM)
Collaboration Level: 1 (https://turing.bowdoin.edu/dept/collab.php)

## Submission Instructions

Use Blackboard to submit the whole project in a zipped format. Include a "readme.txt" file that says how to run your program. You may include some information about the compiler/IDE you have used. We prefer a program that takes the input file name as a command line argument.

- *Unlimited submissions* are allowed. Only the last submission will be graded.
- You may submit the whole assignment by the Part I deadline.

## Programming Languages

Use Python 3. You may use C, C++, or Java, but do so if you are an expert in it.

### Tasks

Build (1) a type system and (2) a semantic analyzer for the CLite language. To do these, you should build on your syntactic analyzer (Assignment 2). Make necessary corrections after you get feedback on Assignment 2. As you can imagine, the inputs here would be a stream of tokens and lexemes. However, recall that lexemes were not at all used in syntactic analyzer. In this assignment, lexemes would be critically important—so important that you will need to use some specialized data structure (i.e., symbol table) to handle them.

In this assignment, some simplifying assumptions will be made regarding the source CLite program. There will be no global variables. There will be just a single function—the main() function. Also, within the main() function, variables declarations will precede all the other statements. Therefore, even if we consider static scoping (as in C, Java, or Python), a single symbol table with just one dictionary will be sufficient for this assignment.

## Part I: Type System (10 points)

The job of a type system is to detect type errors. Typically, a type error means that an operator and its operand(s) are not "compatible." This brings up the question of what is compatible and what is not. Programming languages are remarkably divergent on this issue. For example, the expression `"abc"*2` will not generate a type error in Python, but it will in C. Other common type errors are using an undeclared variable, "incompatible" implicit or explicit type conversion, etc. A type system begins with a list of type rules that specify these rules of compatibility.

In this assignment, you will implement the following type rules.

**Type Rule 1: All variables must be declared before use**

To implement this, you basically need to store some information about the variables declared in the "Declarations" part. One possibility is to build a globally accessible "symbol table" data structure. A symbol table can be implemented by a hashmap (in Java), map (in C++ STL), or dictionary (in Python). Each entry of this symbol table is a key-value pair. Here, each key is a variable name and its value is some information about the variable, such as its type and its R-value. Note that against each key (i.e., variable name), you will need to store multiple things (type and value).

Whenever you encounter an identifier in other parts of the syntax tree (e.g., in an assignment statements or in an expression), you can look up the symbol table and check whether the identifier is there. If not, you will generate a "variable not declared" error.

**Symbol Table Implementation**

*Look at the Codes section of Blackboard for sample codes.*

- In Python, you can use a dictionary to implement the symbol table. The key would be the variable name and the value would be the type of the variable and its value. You can store these two in a (heterogeneous) list. See the sample Python code.
- In Java, you can use HashMap (this is the recommended way among multiple alternatives). The key will be a String object (i.e., variable name). For the value, define a class whose data members are a String (to store type) and an Object (to store value). See the sample code under the Codes section of Blackboard (not under this assignment).
- In C++, you can use STL's map (another alternative would be unordered_map). The key would be a string object (i.e, variable name) and the value of the map would be an object of a customized class that can hold values of multiple types. See the sample code under the Codes section of Blackboard (not under this assignment).

**Type Rule 2: No two variables can have the same name**

You can use the symbol table to enforce this rule. Whenever a new variable is declared in the "Declarations" part, check if there already exists a variable with the same name.

**Type Rule 3: Narrowing conversions are <u>not</u> allowed; widening conversions are allowed**

*You can do it only after implementing Part II: Semantic Analysis!*

In an assignment statement, the left-hand-side variable must be big enough to hold the value generated by the right-hand-side expression. You will implement this rule only for integer and float data. Mixture of Boolean or character data with integer or float will <u>not</u> be allowed in any expression. In those cases, you will detect a type error and exit.

Example: if x is an integer variable and y is a float variable, then none of the following two statements will be allowed by the type system.

```
x = y + 2;
x = 10.0;
```

However, the following will be allowed.

```
y = x + 2;
```

To enforce this rule, each expression must also have a type, which is the "largest" type present in the expression. For example, the type of the expression `y + 2` is float, because y is a float and 2 is an integer. The main challenge in this part is to implement the type of expressions in order to enforce this rule.

**Hint:** The type of an "Expression" depends on the types of "Conjunctions" in it. The type of a "Conjunction" depends on the "Equalities" in it, and so on. Finally, the type of a "Factor" is the type of an identifier or a literal (such as an int, float, or bool) or the type an Expression in parenthesis.

**Suggestion:** You can start your implementation with a stricter version of this rule, namely, no type mismatch is allowed (e.g., for the expression x + y, both x and y have to be of same type). You can then relax it allow widening type conversions.

## Part II: Semantic Analysis (10 points + 1 point extra credit)

In this part, you will keep track of the "state" of the input program from the beginning to the end. The main challenge is evaluating an expression. For example, the value of the expression (2 + 3 * 4) is 14, whereas the value of 2 + 3 * 4 > 0 is True. Most of the statements will depend on the correct evaluation of expressions.

For this assignment, you can expect the following types of statements—assignment, if, print, return (beware—the input will be tokens and lexemes, not a program like the one below). The if statement will have only one statement in its body (no braces). As an example, your program should output 10 for the following CLite program. Here, the output 10 is due to the `print i;` statement. To print the value of the variable i, you will need to look up the symbol table.

```
int main()
{
      int i;
      int k;
```

```
    k = 5;
    i = 0;
    if (k > 0)
        i = 2 * k;
    print i;
    return k;
}
```

**Common Mistake:** Consider the Boolean expression k > 0 within the if statement above. The most common mistake is while evaluating such a Boolean expression changing the value of k. Here, k must not be changed! In fact, the state of the program must remain unchanged while evaluating any expression, be it Boolean or arithmetic! Assignment statements, of course, will change the state of the program.

**Extra Credit: 1 point for while loops (without braces)**

The body of the while loop will contain just one statement (no braces). As long as the condition of the while loop is true, you will have to evaluate the body and make changes to the state of the program. For the following source program, the output of your program should be 16.

```
int main()
{
    int i;
    i = 1;
    while (i < 10)
        i = i * 2;
    print i;
    return i;
}
```

**Input and Output:** The input is a text file with a list of tokens and lexemes in it (tokens in the first column, lexemes second) that came from a syntactically correct CLite program. The output of your program should be type errors (if any) plus outputs generated by print statements. Outputs should be shown on the screen. If there is a type error, your program should exit right away.

**Sample Input (input.txt file on Blackboard):**

| | |
|---|---|
| type | int |
| main | main |
| ( | ( |
| ) | ) |
| { | { |
| type | int |
| id | var1 |

| | |
|---|---|
| ; | ; |
| type | float |
| id | var2 |
| ; | ; |
| id | var1 |
| assignOp | = |
| intLiteral | 50 |
| ; | ; |
| id | var2 |
| assignOp | = |
| floatLiteral | 10.0 |
| ; | ; |
| if | if |
| ( | ( |
| id | var2 |
| relOp | < |
| intLiteral | 2 |
| multOp | * |
| id | var1 |
| ) | ) |
| id | var2 |
| assignOp | = |
| ( | ( |
| intLiteral | 100 |
| addOp | + |
| intLiteral | 2 |
| multOp | * |
| id | var1 |
| ) | ) |
| ; | ; |
| print | print |
| id | var2 |
| ; | ; |
| return | return |
| id | var2 |
| ; | ; |
| } | } |

**Sample Output:**

200

**Design Issues:**

This assignment leaves some design questions open. For example, what are compatible types for arithmetic operations? What will be the result of an integer division (such as 3/2)—will the result be float (such as 1.5) or int (such as 1)? Can a Boolean value be added to an integer

resulting in an integer value? For simplicity, we will assume that you cannot add a Boolean value to an integer number, even though it's allowed in C. The main requirement here is that a float variable can be assigned integer value due to widening conversion. If you make additional design decisions, make sure that you write them down in your readme.txt file.

**About the Sample Python Program *semantics.py*:**

The sample program implements a minimal type system and semantic analyzer for CLite programs with a very restricted grammar. There is also a sample input file on Blackboard, *python_input.txt* for this program. This input file (*python_input.txt*) is only for this sample Python program (*semantics.py*) and is not an input for this assignment.

Here are some points regarding what the sample Python program does and what its limitations are.

1. For the type system, it implements the type rule that a variable must be declared before being used. It shows how to build a symbol table using a dictionary in Python and how to access elements of that dictionary.
2. For the semantic analyzer, the sample program shows how to evaluate an expression, where the expression is evaluated to an integer. For this assignment, you will need to consider other possible types in addition to integer, such as float and bool. Also, if you look at the grammar used in the sample program, you will see that an expression does not have any identifier in it; it only has integer literals.