CS 423 Final Examination **Name _____**

Answer all questions. This is an closed book, no electronics, open note (up to 30 sheets of paper) exam.

1. (20 points) In mainstream languages, to test whether a value `x` is in the range between two other values `low` and `high`, you would write `((low < x) && (x < high))`. In this case, the result of the less-than operator is a boolean (or a 0 or 1 in C/C++). In contrast, Python allows you to write the simpler and more readable notation `(low < x < high)`. Is this feature in Python lexical, syntactic, or semantic? What is the result type produced by the less-than operator in order for this notation to be legal? Did you have to support this feature in PunY? If so, how did you, and if not, how might you go about it?

2. (10 points) There were some issues in the PunY test programs, where the name `string` was used instead of `str` to denote the string type. For example, consider the (attempted) variable declaration

```
s : string
```

If I saw your compiler printing a syntax error message saying the token `ing` was appearing in an illegal location in the code, on a line where the only occurrence of `ing` was in an instance of `string`, what problem is the compiler having, and how might it be fixed?

3. (20 points) Write a regular expression for Python lists of integers constructors consisting of an open square bracket followed by 0 or more comma-separated integers, followed by a closing square bracket. Why might the Python language designers choose to form list constructors out of many tokens instead of resolving them entirely with a regular expression like this?

4. (10 points) Near the start of this semester, students were given a grammar for Python that came directly from the “real” C Python 3.x source code distribution that can be understood to be the official Python language grammar. Why was that grammar not that great for our project? What changes did you have to make to that grammar in order for us to use it?

5. (10 points) Consider a recursive context free grammar rule in Bison, such as

E : E + E ;

True or False:

5.a) The production rule is left-recursive.
5.b) The production rule is right-recursive.
5.c) The production rule is written in syntactically correct Bison format.
5.d) The production rule is logically complete and usable as-is.
5.e) The production rule can produce many instances of E chained together by plus operators.

6. (10 points) Fill in the blanks.

Populating a symbol table is performed using a _____ algorithm that visits the nodes in the syntax tree. The leaf nodes consisting of _____ are inserted into the symbol table whenever the surrounding context indicates that a _____is being declared.

7. (20 points) Multiple choice. Circle the correct answer(s) to the following question, if there are any.

What is the `.place` semantic attribute used for in code generation as taught this semester?

a) `.place` stores the location of a tree node within the syntax tree.
b) `.place` indicates the line number where a leaf occurred within the source code.
c) `.place` gives the order competing instructions go during a race condition.
d) `.place` contains the GPS locations where code was originally written.

8. (30 points) In this semester we did not implement PunY class syntax, but we did need to support the syntax `x.f(params)` where `x` is a Python value, `f` is a method name, and *`params`* are zero or more parameters for that method. Why did we need to support this syntax? Describe a plausible method by which a compiler might implement this syntax for built-in methods of built-in types in the language.

9. (20 points) Analyze the Python code fragment below. Draw syntax trees for the executable statement(s). Report what a PunY compiler's type checker would do in order to determine whether the types were correct. Then report what the outcome of PunY type checking would be.

```
def main():
    j : int
    j = input("Enter a number ")
    print("It was: ",  j - 2)
main()
```

10. (20 points) Draw a syntax tree for the following Python code. Generate intermediate three address code (in the form of a linked list diagram) for them.

```
def main():
    j : int
    j = 0
    while j < 3:
       print(“j2: ”,  j – 2)
       j = j + 1
main()
```

11. (20 points) In this semester you had a choice between generating VM bytecode (Ucode) or writing a transpiler that generates Unicon source code. In most respects, the transpiler option would be easier. Describe technical reasons why generating VM bytecode might be easier than a transpiler for some languages and/or some language constructs.

12. (20 points) What are the primary tasks involved in final code generation? If you had had to write a compiler that generates native x86_64 code for PunY, what code generation strategy would you use for language constructs such as dictionaries (dicts) that are not built-in to the x86_64?