COSC4315/COSC6345: Checking Data Types

1 Introduction

You will create a "compiler" program to detect data type conflicts in source code. The input source code will be: Python. This "interpreter" program will be developed in C++. The output will be a 2nd program inserting comments with warning/error/undefined messages.

Your program will use regular expression to recognize identifiers, numbers and strings. Your program will use a simplified context-free grammar to recognize arithmetic expressions with '+'. In order to detect data types you will have to perform a partial evaluation using an attribute grammar to extend the parse tree. Your program does not have to generate intermediate or object code.

2 Input

The input is one source code file. The programs will contain the following statements: assigment with arithmetic expressions, function calls, if/else statements, function definitions with up to 3 arguments. The main arithmetic operator will be the '+' and '*' operator. The variables can be numbers, strings or lists. In the case of numbers + means addition, for strings it means concatenation and for lists union. Notice '*' is not available for strings or lists. The if condition will be one comparison (no and/or).

You can assume the program will be syntactically correct: detecting syntax errors will be tackled in another assignment. You can assume the input source code has no classes, no loops and no recursive functions. You can assume there will be no function calls to convert data types (casting).

3 Program and output specification

The main program should be called "checkdatatype.cpp".

Your program will be compiled:

```
g++ checkdatatype.cpp -o checkdatatype
```

Call syntax from the OS prompt (rename a.out!):

```
# if in path or ~/bin
checkdatatype file=program1.py

# default
./checkdatatype file=program1.py
```

3.1 Input and Output:

You will be given a syntactically correct python program as your input file ($program1.py$). Your output should be in *input_filename*.out (ex: $program1.out$) which will print "#warning" or "#error" before the line. For correct statements, you do not have to print anything.

4 Requirements

- You should store all the identifiers in some efficient data structure with access time O(1) or O(log(n)). These include variable and function names.
- The arithmetic expression can have up to 10 operands combining + * () and function calls.
- The input is one .py file and it is self-contained (this file will not import other py files). The output is a 2nd .out file with more comments.; this 2nd file should work exactly like the input file.
- In Python a list can mix data types, but in this homework we will take a stricter approach by displaying a warning when at least one element in the list has a data type conflict with respect to the other ones.
- Your program should write a comment before each line: error, warning, undefined, following Python semantics. In order to make the output shorter do not display any message when the line is correct. ”undefined” happens when it is not possible to determine data types (for instance, if there is a variable provided by the user or an argument to a function that is not called).
- It is acceptable to have one variable instance, overwriting the previous ocurrence. That is, you do not have to create new objects.
- You cannot use an existing Python parser. You have to build your own.
- You should use and explore the Python interpreter to verify correctness of your program. Keep in mind this homework is asking you to develop a ”stricter interpreter” that performs more static data typer checking than the standard Python interpreter.
- The program is required to detect data type conflicts. The program does not have to evaluate the Python expressions to produce a result.
- You have to create a ”binding” data structure to track data types; which must be clearly highlighted in your readme file.
- There will not be ”cast” or type conversion function calls since that would require to track types in functions.
- The program should not halt when encountering syntax or data type errors in the input source code.
- Optional: For each variable you can store its data type and a list of lines where it was set or changed.
- Your program should write error messages to a log file (and optionally to the screen). Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.
- Test cases: Your program will be tested with 10 test scripts, going from easy to difficult. If your program fails an easy test script 10-20 points will be deducted. A medium difficulty test case is 10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.
- You can assume the given python program will be clean and there will be no syntax error. 70% of the grade for each test case will be to detect errors and 30% of the grade will be for detecting warnings. Your program must shows error where python shows error.

- A program not submitted by the deadline is zero points. A non-working program is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 80 or higher. In general, correctness is more important than speed.
- Your program will be initially executed using an automated script. So, make sure you follow the filename and syntax format given in Section 3.