CS403: Programming Languages

The Designer Programming Language

Version 1

Preliminary information

Your task is to build an interpreter for a general purpose programming language of your own design. Your language must support the following features:

- comments
- integers and strings
- dynamically typed (like Scheme and Python)
- arrays with O(1) access time
- conditionals
- recursion
- iteration
- convenient means to print to the console
- an adequate set of operators
- anonymous functions
- functions as first-class objects (i.e. functions can be manipulated as in Scheme - e.g. local functions)
- (graduate only) strong typing (type errors detected at parse time)

The only basic types you need to provide are integer and string and you do not need to provide methods for coercing one type to another (although you may find it convenient to do so). The efficiency of your interpreter is not particularly important, as long as you can solve the test problem in a reasonable amount of time. Your language also does not need to support reclamation of memory that is no longer needed. You are to write your program in a statically-typed, imperative language such as C, C++, or Java. Check with me first if you wish to use some other host language.

Test Problem

You are to write a program to implement the Sieve of Eratosthenes, starting with a (potentially) infinite stream of numbers. Provide a README file that explains how to run and test your program. Your program should not use assignment.

You will receive a serious deduction if any rule in your makefile causes a pause for input.

Grading

- [100 points] everything works
- [51-99 points] functionality is missing/test program is missing
- [50 points] pretty printing
- [30 points] recognizing

Extra credit will be given to exceptional implementations. Your README file should give pertinent details.

For Undergraduates: if you do not, at least, implement a recognizer for your language, you will fail the course.

For Graduates: if you do not, at least, implement an pretty printer, you will fail the course.

Submitting the assignment

To submit your designer programming language, place all your source code, sample programs, a README detailing how to run and write programs in your language, and a makefile for building your system into one directory. Name the README file *README*. Your makefile should respond the command

```
make
```

which builds your processor and to the following commands, each of which illustrates a feature of your language:

```
make error1
make error1x
make error2
make error2x
make error3
make error3x
make error4
make error4x
make error5
make error5x
make arrays
make arraysx
make conditionals
make conditionalsx
make recursion
make recursionx
make iteration
make iterationx
make functions
make functionsx      # shows you can pass functions and return nested functions
make lambda
make lambdax
make dictionary
make dictionaryx     # shows that you have a log(n) AVL-tree dictionary
```

The first rule in a pair of rules should print out the appropriate input program, while the x rules should execute that program. In particular, the first three error rules should show off your parser detecting three different kinds of syntax errors, while the last to error rules should demonstrate the detection of two different kinds of semantic errors.

Your makefile should also respond to the commands:

```
make problem
make problemx
```

These commands display the test problem of your implementation and run the test problem, respectively.

Finally, provide an executable shellscript named *dpl* that runs a program like so:

```
dpl testprogram1.mylang
```

Test programs can be named anything.

Note: in the case of recognizing only, only the error rules need be present in your makefile. In the case of pretty printing only, the *run* rules should run the input program through the pretty printer.

Finally, your makefile should respond to the command:

```
make clean
```

This command should remove all compilation artifacts (such as `.o` or `.class` files) so that a clean compile can be performed.

Makefiles for graduate students should additionally respond to the commands:

```
make grad
make gradx
```

These rules should thoroughly demonstrate the additional requirements for graduate students.

To submit your language interpreter, run the command:

```
submit proglan lusth dpl
```