Cesium : A New Language

William B. Hart

November 23, 2012

Contents

1 Introduction

Cesium is a new programming language designed primarily for efficiency, easy interface with C libraries and with a straightforward high-level syntax. It uses the LLVM Jit to give very performant code, even when run in interactive mode. The language is polymorphic, statically typed (with type inference) and imperative in style, with some functional features.

The language fills a niche between Python and C. Python is not terribly performant, but has a very simple and flexible syntax. C is fiddly and low level, statically compiled and not interactive, but very fast.

Cesium is inspired by Julia, but is more suited to computer algebra, rather than numerical tasks. Of course Cesium can be used as a general purpose language, but design decisions are influenced by the desire to make it useful to a certain class of mathematical/computational people.

2 The Cesium interpreter

The Cesium interpreter is started by typing `./cs` at the command line.

When the interpreter starts, a prompt is displayed ready to accept Cesium code.

If you type an expression and press enter, Cesium immediately executes the expression and a return value is displayed. If there is no return value, `None` is printed.

Note that end-of-lines are significant in Cesium. In many cases they can be elided by addition of certain keywords. However, in general, statements must finish with an end-of-line.

3 Literals

We first describe how to enter various types of constant literals in Cesium.

3.1 Integer literals

Here are some examples of signed and unsigned integers in Cesium:

```
1234
12u8
109i8
278009804u
234567i32
```

Integers with no suffix are signed and either 32 or 64 bits, depending on whether a 32 or 64 bit binary of Cesium is being used.

Integers which are unsigned are given the suffix `u`.

The other suffixes, e.g. `i8`, `u32`, etc., specify whether the number is signed (with a `i`) or unsigned (with a `u`) and how many bits the type is. The valid numbers of bits are `8, 16, 32, 64`.

For example, the literal `109i8` is the integer `109` stored as a signed 8 bit value.

The suffixes are necessary in Cesium because it uses type inference to infer the type of a value from the value itself. In the case of numbers this is not possible without the extra annotation.

In practice, only unannotated types are required unless one has some specific bit fiddling to do or if one is interfacing with a C program which makes use of specific types.

3.2 Floating point literals

There are two types of floating point values in Cesium, `float` and `double` values.

Floating point values must have a decimal point and have an optional exponent. The exponents begin with an optional sign, followed by an integer.

Without a suffix, floating point values are considered to be double precision floating point values. However, if they have the suffix `f` they are considered to be single precision floats.

Here are some example floating point values:

```
1.0
2.1f
1.3e+12
1.4E-11f
```

For example, `1.4E-11f` is the single precision floating point value 1.4×10^{-11}.