Physicalc:
A Language for (simple) Scientific Computation

Brian Foo, bwf2101@columia.edu
Changlong Jiang, cj2214@columbia.edu
Ici Li, il2117@columbia.edu
Stuart Sierra, ss2806@columbia.edu

Project Proposal – September 25, 2007

1 Introduction

Physicalc is a programming language for scientific computation, designed for students studying beginning and intermediate-level physics, chemistry, or other sciences.

Computer algebra systems are typically oriented towards higher mathematics, making them ill-suited to the sorts of calculations done by high-school and undergraduate science students. At the same time, some computer algebra features, such as symbolic computation using irrationals, could be helpful to students. Physicalc presents itself initially as an intelligent calculator that understands physical units like "meters/second." It can also solve simple problems involving physical equations. For more advanced users, it supports real programming in an imperative style.

Physicalc is intended primarily as an educational tool, but may also be useful for exploratory data analysis in scientific fields.

2 Language Overview

2.1 Interpreter

Physicalc is an interpreted programming language. The interpreter is written in Java and can be run either interactively or on a file containing Physicalc source code. The interface is text-mode, although a GUI could be layered on top of it.

2.2 Syntax

Physicalc syntax is as simple as possible, using mostly English words, more reminiscent of BASIC than C. Statements are separated by newlines. Statement blocks are enclosed in "do...done" pairs. Standard imperative-language

features such as loops, if/then/else branching, and user-defined functions are provided. Standard mathematical operators are provided, with the addition of '^' for exponentiation.

Identifiers are both case-insensitive and inflection-insensitive. That is, 'newton', 'newtons', 'Newtons', and 'NEWTON' are all the same identifier.

2.3 Types

The Physicalc type system uses physical units like "meters" and "seconds" as types instead of mathematical partitions like "integers" and "floats." Unit types can be combined algebraically to form derived types such as "Newton*meters" or "meters/second."

All numbers are arbitrary-precision decimals or rationals. Limited symbolic computation is supported—irrational numbers such as π and $\sqrt{2}$ can be used in calculations and returned in results, or converted to decimals with an arbitrary degree of precision. Complex numbers are supported.

Two-dimensional vectors may be written either as "x, y" components or as magnitude-direction pairs, e.g. "3 Newtons at 36 degrees." Directions given as angle measures are assumed to be measured counterclockwise from the x-axis.

The following operations are supported on all types: addition, subtraction, multiplication, division, exponentiation, and roots.

2.4 Semantics

A Physicalc program consists of definitions, queries, and expressions.

2.4.1 Definitions

1. *Quantities* define the types of measurements that can be made. These may be fundamental quantities, such as length, or derived ones, such as momentum. Quantities can be defined in terms of other quantities. Examples:

   ```
   def quantity distance
   def quantity time
   def quantity velocity = distance / time
   ```

2. *Units* define units of measurement for a quantity. Units may be defined in terms of other units. Examples:

   ```
   def unit meter for distance
   def unit centimeter = 0.01 meters
   def unit foot = 30 centimeters
   ```

3. *Aliases* define alternate names for quantities or units. They may be used for abbreviations or alternate names. Examples:

```
def alias length for distance
def alias N for Newton
def alias feet for foot
```

4. *Constants* are fundamental physical constants that can be used as quantities in equations. Example:

```
def constant universal_gravitation = 6.67428e-11 N*m^2/kg^2
```

5. *Equations* define algebraic relationships between quantities. They are written as expressions followed by a series of declarations describing the quantities of each variable. Example:

```
def equation Fg = G * m1 * m2 / r^2 where
    Fg = gravitational_force
    G = universal_gravitation
    m1 = mass
    m2 = mass
    r = distance
done
```

6. *Functions* are standard imperative-style functions, which may be recursive. GCD in Physicalc looks like this:

```
def function gcd(a,b)
    while a != b do
        if a > b do
           set a = a - b
        else
           set b = b - a
        done
    done
    return a
done
```

2.4.2 Queries

Queries ask questions of the form "find *unknown* given *knowns*." Example (syntax to be determined):

```
find mass in pounds given
    force = 12 Newtons
    acceleration = 3 m/s^2
done
```

Queries work by exploring the graph of relationships among physical quantities and units defined in a program. Based on the types of the givens and the requested type, the interpreter infers the correct sequence of calculations and/or conversions to be performed.

2.4.3 Expressions

Expressions are any combination of mathematical operators and function calls.

3 Prior Art

3.1 Google Calculator

Superficially, the behavior of Physicalc resembles that of the Google Calculator[3], which can answer queries like "160 pounds * 4000 feet in Calories." The differences from the Google Calculator are:

1. Physicalc is a full-featured imperative programming language with variables, loops, branching, and user-defined functions;
2. Physicalc programs can define new units and the relationships among them; and
3. Physicalc can infer the necessary calculations to obtain a desired result given a set of known variables.

3.2 Computer Algebra

Physicalc has some features in common with computer algebra systems such as Maxima[8], Octave[2], and MATLAB[7], although it is much simpler. As far as the authors know, none of those systems allow for unit types or inferred calculation. Also, most such systems are oriented towards higher mathematics and are far too complex to be useful to beginning science students.

Various Java libraries [1, 4, 6] provide computer algebra features; these may be useful in the implementation of Physicalc, probably with some modification.

3.3 Logic

In some ways Physicalc behaves like logic languages such as Prolog, in which the user enters a series of facts and then enters queries about those facts. However, Physicalc is not a general-purpose logic or constraint programming language.

3.4 JScience

JScience[5] is an open-source Java library for calculations involving SI units and arbitrary-precision arithmetic. JScience's unit classes are of limited usefulness within Physicalc because they are designed to be part of a statically-compiled

Java program. Also, JScience does not provide for inferred calculation. However, the JScience architecture is a useful model for designing Physicalc; and some features of JScience, such as its arbitrary-precision arithmetic and algebra classes, may be useful in the implementation.

4 Extras

The following features may be added given sufficient time and ambition.

- Three-dimensional vectors
- Pretty-printing expressions in plain text
- TEX output
- GUI interface
- Maximizing/minimizing functions
- Two-dimensional and solid geometry
- Plotting
- Simulation of physical systems
- Hash tables
- Objects
- More advanced algebra (trigonometric identities, factoring polynomials)

References

[1] Apfloat, `http://www.apfloat.org/apfloat_java/`

[2] GNU Octave, `http://www.gnu.org/software/octave/`

[3] Google Calculator, `http://www.google.com/help/calculator.html`

[4] Java Algebra System, `http://krum.rz.uni-mannheim.de/jas/`

[5] JScience, `http://jscience.org/`

[6] Jscl-meditor, `http://jscl-meditor.sourceforge.net/`

[7] MATLAB, `http://www.mathworks.com/products/matlab/`

[8] Maxima, `http://maxima.sourceforge.net/`