PROPOSAL: A Stack Game in a Stack Language

Reggie Gillett
27133123
w6k8
reggie.gillett@gmail.com

Graham St-Laurent
23310121
i5l8
gstlaurent@gmail.com

Lynsey Haynes
12686119
a4h8
lynseyahaynes@gmail.com

Brittany Roesch
22015119
r1d8
brroesch@gmail.com

Gord Minaker
34615112
u4s8
gordonminaker@hotmail.com

ABSTRACT

Factor is a richly-featured stack-based, concatenative, object-oriented programming language. Using this language, which none of us has studied previously, we plan to create a substantial program in the form of an interactive game. This paper provides an introduction to Factor as a programming language, outlines how we will use Factor in our implementation, and explains our approach to meeting all project milestones.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Languages

Keywords

Factor, games, stack-based languages, concatenative languages

1. INTRODUCTION

We will be creating a substantial program in Factor[1] — a concatenative, stack-based programming language. Our game will educate and illustrate to players the unique paradigm of concatenative languages. We intend to create said game in Factor, which is itself a concatenative language. This is not an accident.

2. OVERVIEW OF FACTOR

Unlike most of the languages we've encountered, Factor is both concatenative and stack-based, but it is still rich with features that are found in more typical modern programming languages (E.g., graphical, networking, threading, and I/O capabilities)[2]. This leaves open the possibility of creating a substantial and useful program without too much effort. Factor also implements a diverse collection of functional patterns, leading to potentially intuitive ways to manipulate sequential data[3]. But unlike most high-level languages—functional, or otherwise—Factor provides the option of using low-level features such as manual memory management, pointer arithmetic, and inline assembly code. Although we don't expect that coding our program will take us in that particular direction, this juxtaposition of high and low levels puts Factor in the same camp as Go[4] or Rust[5]—just like C (or perhaps, in Factor's legacy, Forth[6]) was forty years ago—and that under the right circumstances, this could be quite useful (albeit, potentially quite dangerous, too!). Ultimately, "Factor combines features from existing languages with new innovations"[7], and it is through that eclectic mix of tools for approaching problems[8], that we will explore Factor, perhaps leading to surprising solutions, and some of those might turn out to be quite enlightening.

In spite of the diverse approaches to programming that Factor provides, its syntax demands that actual coding will always be stack-based and concatenative. It is the value of this syntax and its unique patterns, that we find the most interesting, and potentially, the most useful. Concatenative syntax is point-free (although Factor does provide identifiers and its creators recommend using them for particularly complex situations[9]), and stack-based languages use postfix operators; both of these syntactical styles are atypical among today's most popular programming languages (with the exception of Haskell, which has a popular point-free style). This means that Factor has a unique way of expressing and exploring, and we will develop a new way of looking at problems that could come in handy in our future endeavours[10]. In the course of working on our program, we will try to use as many of these strange idioms and patterns as we can. The creators of Factor have given

[1] http://factorcode.org

[2] http://docs.factorcode.org/content/article-vocab-index.html

[3] http://docs.factorcode.org/content/article-lists-combinators.html

[4] https://golang.org/

[5] http://www.rust-lang.org/

[6] http://www.forth.com

[7] Slava Pestov, et al, "Factor: A Dynamic Stack-based Programming Language", *ACM Proceedings*, 2010.

[8] Slava Pestov, *Factor/Features/The language*, 2010. http://concatenative.org/wiki/view/Factor/Features/The%20language

[9] Slava Pestov, Factor Philosophy - Factor Documentation http://docs.factorcode.org/content/article-cookbook-philosophy.html

[10] Jon Purdy, *Why Concatenative Programming Matters*, 2012. http://evincarofautumn.blogspot.ca/2012/02/why-concatenative-programming-matters.html

evocative names to many of these; we intend to produce a program turgid (where appropriate, that is) with things like *Fried Quotations*, some of the numerous *combinators* (e.g., *Cleave, Spread, Sequence, Curried dataflow*), and, perhaps most interestingly, functions that assemble or disassemble *quotations* (lambdas), thus providing an idiomatic way to "write code that writes code"[11].

3. OUR APPROACH

The 80% – Background Research Report

To complete this milestone, we will research even more extensively the advantages (and disadvantages) of Factor as a language. Principally, we will describe the concatenative and stack-based language paradigms, how Factor implements them, while comparing Factor's implementation and syntax to those of other stack-based languages (such as PostScript[12]). We will focus on Factor's features that we plan to use in our game, such as: input, output/graphics, control flow, unit testing, and object-orientedness. Throughout this section our theme will be constantly appraising and re-appraising Factor's usefulness.

The 90% – Proof-of-Concept and Plan

In order to create a game, we will need to (1) learn how to code in Factor, (2) make sure that Factor has the tools we need to do what we want to do, and (3) have a game design in mind. In this section, we will demonstrate that all three things are possible. For (3), we will provide a detailed layout for the game that we will complete. It will be our plan, henceforth, to implement every element that we describe in this layout, so it is this at this milestone that the bulk of our game's design will be finally decided. (2) and (1) go hand-in-hand: by fulfilling (2), we will be proving that we can fulfill (1). In order to fulfill (2), we will implement tiny programs that highlight key features that this game will rely on. The features we intend to demonstrate are the following:

- **Display**: We will determine how we will implement the visual component of our game by providing some simple, changing images. These will either be standard graphics, or text-based graphics, if the former turns out to be too troublesome for the scope of our project.
- **Input**: A tiny program to demonstrate that we can dynamically read user input, be it through text, arrow-keys, mouse, or some combination.
- **Control Flow**: Since the backbone of our game will rely on a game loop, we will demonstrate that Factor provides sufficient control flow structures to make a loop, as well as make decisions.
- **Unit Testing**: Robust program development requires reliable and easy testing. Factor provides a unit testing framework, and we will demonstrate that it meets those criteria.
- **Object-Orientedness**: Factor is an object-oriented language. Since there are a number of different ways of implementing the object-oriented concepts, we will demonstrate that Factor's extensible approach is suitable for us by doing some basic object/method manipulations.

Some, if not all, of these features can be demonstrated together in tiny programs, or some may be best shown in multiple programs, so our proof-of-concept may not consist of exactly five programs. It is also possible that, as we get to know Factor better, we will discover other features that we determine to be indispensable for our game, in which case we will demonstrate them here also.

The 100% – Final Project

Our core game engine will be implemented in Factor, including at least one level. Gameplay will be possible using the features that were demonstrated in the *Proof of Concept*, as well as a fully-implemented game-loop. Externally, this will be evident to the user as an image of a stack, a list of commands, and a problem to solve (as well as instructions). The user will be able to select the appropriate commands for either success or failure, and the game will respond accordingly. Internally, this will be accomplished through intended-to-be-idiomatic use of Factor.

The effect that idiomatic use of Factor will have on our code has two main components. One is simply that fact that *we are using Factor*, so all of the standard things that one does while programming will be done using Factor's particular—and sometimes unusual—implementations. In particular, we refer to the features that we will demonstrate in the *Proof-of-Concept*, described below. The other component is rooted in the fact that Factor's tagline (yes, this language has a tagline) is "A Practical Stack Language"[13], so it has developed a number of features to make its concatenative syntax more convenient (such as the aforementioned *combinators*). We will be using these features as well as the stack-based, concatenative syntax in general, and this will affect how we think about the code that we write. Upon completion will make a value judgement on the usefulness of the concatenative paradigm.

The Poster

The poster will have three purposes; these correspond nicely to the three components of the *Proof-of-Concept*, described above. The first purpose will be to introduce people to the likely-unfamiliar concept of stack-based programming and its point-free, concatenative syntax. Some nice stack images will help with this. The second component will feature Factor specifically: we will choose one or more particularly-interesting features of Factor (likely, these will include a unique solution to a common concatenative problem—e.g., readability); we will then display some code that exemplifies this, and have some stack-based, graphical explanations of them. Finally, there will be a small component that describes the game that we made: our peers will be able to see some real Factor code from a real program! In fact, by then, enough of the game will have been completed that we could potentially have a computer set up and provide a small demonstration of our Factor-coded game for our peers.

4. STARTING POINTS

[11] Slava Pestov, Control Flow Cookbook - Factor Documentation, 2014. `http://docs.factorcode.org/content/article-cookbook-combinators.html`

[12] `http://www.adobe.com/products/postscript/`

[13] `http://factorcode.org/`

We have found several scholarly articles and online resources which will aid in the development of our project:

- Pestov, S., Ehrenberg, D., Groff, J.: *Factor: a dynamic stack-based programming language.* In: DLS 2010 Proceedings of the 6th Symposium on Dynamic Languages (2010)

  A paper which presents Factor as a language written by its creators. (Note the reference to our own Gregor Kiczales in the bibliography)

- Herzburg, D., Reichert, T.: *Concatenative Programming: An Overlooked Paradigm in Functional Programming* In: Proceedings of the 4th international conference on software and data technologies (2009)

  A paper which espouses the value of concatenative languages in software engineering research. The authors present a language they have developed called Concat which exemplifies the unique qualities of the paradigm.

- `http://concatenative.org`
  in particular:
  `http://concatenative.org/wiki/view/Factor`

  Concatenative.org, a wiki dedicated to concatenative languages.

- Slava Pestov (October 27, 2008). Factor: An Extensible Interactive Language (flv) (Tech talk). Google.

  A talk which describes the rational for Factor's creation, an overview of the language and its features, and a demonstration of how it can be used.

- Zed Shaw (2008). The ACL is Dead (flv) (CUSEC 2008). CUSEC.

  A presentation written in Factor which mentions and praises Factor

- The creators of Factor have also provided numerous resources on their webpage:

  `http://factorcode.org/`

5. SUMMARY

We are creating a substantial program in Factor; a programming language that none of us have studied previously. Factor is a concatenative, stack-based programming language and we will be using it to make a game which, correspondingly, helps to teach users about stack-based languages.