Bridge: Development of an Intermediary Programming Language

BOSTON UNIVERSITY

Vidhu Nath

Advisor: Andrei Lapets, Ph.D., The Rafik B. Institute for Computing and Computational Science

Abstract

This Keystone Project is the creation of a small, contained, and functional programming language that can be used to introduce non-technical users to the concepts of computer science and programming. It has three main goals:

- Create a language is user friendly
- Develop the tokenizer, parser, and interpreter components of the language
- Spread simple understanding of programming concepts.

With these three goals, Bridge has been designed with user interaction in mind, and aims to provide users with the confidence to use programming languages.

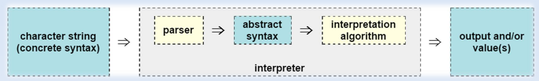
Objectives

The purpose of Bridge is to provide a means of bringing programming understanding to a population that otherwise might avoid such a pursuit. The goal is not to produce many high-level programmers, but to excite every day users about the possibilities and potential within programming. By including only a limited functionality, Bridge encapsulates the ideas of programming without overwhelming users with features.

Components

There are three main components of Bridge: the *tokenizer*, the *parser*, and the *interpreter*. The general structure of a programming language is as below, courtesy of Andrei Lapets:

Below is a more comprehensive representation:



Consider the following input, which is called *concrete syntax*:

x = 1; print x

The tokens produced by the *tokenizer* on this input would be:

[“x”, “=“, “1”, “;”, “print”, “x”]

These tokens would be read into the *parser*, which outputs the following *abstract syntax*:

```
{
  "Assign": [
    "x",
    {
      "Term": [
        {
          "Factor": [
            {
              "Number": [
                "1"
              ] } ] } ]
    },
    {
      "Print": [
        {
          "Term": [
            {
              "Factor": [
                {
                  "Variable": [
                    "x"
                  ] } ] } ] } ] }
```

The *interpreter* takes this result to output the result of running the code:

1

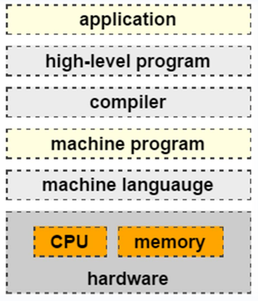
This is the output of the “print” on the variable “x”.

Background

The foundation of a programming language is the series of *low-level instructions* it provides the hardware. These instructions take form as *assembly language*, which directly interacts with the machine. Below is an example of assembly:

```
# The Fibonacci sequence
# Uses a separate function to print the first n Fibonacci numbers,
# and to compute a print their sum.
#
# We assume that n >= 2.
#

00   read r1             # r1 stores n
01   call r14 04         # call the function, storing return addr in r14
02   write r13           # write the sum (the return value)
03   halt

# the beginning of the separate function
04   setn r2 0           # r2 stores the last fib # (initialized to 0)
05   setn r3 1           # r3 stores the current fib # (initialized to 1)
06   setn r13 0          # r13 stores the return value (the sum, init to 0)

07   jeqz r1 15          # stop once n is 0

# print the current number, add it to the sum, and compute the next one
08   write r3            # print the current number
09   add r13 r13 r3      # add the current num to the sum
10   add r4 r2 r3        # compute the next fib # (r4 = r2 + r3)
11   copy r2 r3          # the current fib # is now the last one
12   copy r3 r4          # the next fib # is now the current one

13   addn r1 -1          # n = n - 1
14   jumpn 07            # go back to loop test

15   jumpr r14           # return from function call
```

Programming languages are the basis for any software, from Facebook to MacOS to video games. Any *application* which relies on a programming language can be represented as below:



The combination of the *high-level code* typed with the *low-level instructions* is what produces all of the software that exists.

Language

A programming language and its functionality can be represented by its *Backus-Naur Form* (BNF). Below is the BNF for Bridge:

Program ::=
Assign | assign variable := expression ; program
Print | print expression
For | for term times program ; program
If | if formula program ; program
End | end

Expression ::= term | formula

Formula ::=
And | formula and formula
Or | formula or formula
Not | not formula
Greater Than | term gt term
Less Than | term lt term
Greater Than/Equal | term ge term
Less Than/Equal | term le term
Equal | term eq term
True | true
False | false

Term ::=
Plus | term + term
Minus | term – term
Factor | factor

Factor ::=
Mult | term * term
Div | term / term
Power | term ^ term
Mod | term % term
Number | (0[1-9][0-9]*)
Variable | [a-z][A-Za-z]*

Acknowledgements

I would like to thank Professor Andrei Lapets for being my advisor; Eric Dunton for being a valuable resource; Rachel Costa and Lauren Mattera for testing my language. Also a very special thank you to Charles Dellheim and the entirety of Kilachand Honors College and Boston University.