bJou

Programming Language

Brandon Kammerdiener

An introduction to the language, its design, and motivation through example.

Contents

Goals

Goals of this Document

This document aims to provide background and explain the motivations behind the bJou programming language. It will also describe and demonstrate the features of bJou by guiding the reader through code examples that can be compiled with the bJou compiler that goes with this document.

Goals of this Language

bJou is my attempt to create the programming language that I want to use. So, its features and design are almost entirely based around my specific needs and interests. I love lower-level programming. Stuff like, well, compilers. Things that are important to me in a language are direct access to memory and hardware, performance, and expressiveness. That list is probably not too surprising and truthfully, there are many languages that take these priorities and are great languages. C is incredibly fast. Writing Python is like writing poetry. bJou seeks to take what are, in my opinion, the best attributes from many languages and combine them into one. In short, bJou is a compiled, statically typed, multi-paradigm language with an emphasis in clear and intentioned abstraction techniques. bJou also takes an interesting approach to metaprogramming, which will be explored later.

Setup

Getting bJou

Setting Up Your Environment

The Language

Now that everything is up and running, we can look at some specific examples of what the language is and what it can do. One important thing to mention before we continue is that many syntax choices of the language in its current state are temporary and will most likely change. The features and ideas are more important at this point anyway. Onwards!

Variables, Type Intelligence

```
# demo1.bjou
# Variables, Type Intelligence

proc main() {
    num : int
    num = 12345
    word : char* = "Foo"

    new_num : int* = new int

    floatingpt := 56.789
    new_char := new char

    @new_char = 'b'

    print "num: %, word: %, floatingpt: %, new_char: %", num,
↪   word, floatingpt, @new_char

    delete new_num
    delete new_char

    printf("%c\n", "string"[3])

    printf("%f\n", { 1.23, 4.56, 7.89 }[1])

    # array0 : int[num]
    array1 : int[3 + 2]
    array2 := { 1, 2, 3, 4, 5 }
    # i := array2[1+6]
    print "array[3] = %", array2[1+2]
}

main()
```

Code