article
geometry margin=1in listings hyperref
language=Bash
2in The Artifice Programming Language 3in Noé Garcia June, 2024
document
Introduction to Artifice

Artifice is a small functional programming language developed as a hobby project starting in 2023. Artifice is built to be a bridge between imperative and declarative functional programming language domains. Built to be simple and easy to read and write with, Artifice is a statically typed, semi-pure functional language with basic support for simple tooling and functionality. While in its infant stages, the language will grow to support more common aspects found in other languages.

Functional programming can be intimidating for those unfamiliar to many who have never worked with it before. For this reason, Artifice is a case study for a programming langauge which aims to make the introduction to the functional paradigm more familiar and easily approachable.

Artifice is a general programming langauge as constructed, but domain specific library extensions will be built to extend the capabilities of Artifice into specific use cases. As it stands, there are plans to epand on Artifice by adding the ability to write logic centric programs.

Artifice is built to take advantage of the different types of declarative programming paradigms such as function and logic programming. All features are experimental and may be subject to change.

This text is an introdution to the Artifice language as it is, and should be used by anyone who is looking to learn how to read and write in the language.

Functional Programming

What is functional programming? In essence, functional programming is the practice of building computer programs using functions. While this sounds similar to other languages if you have worked with languages such as C or Python, there are a few rules that the functions in functional programming languages follow that are not followed in other imperative langauges.

The first is that functions should all accept at least one variable. And, each function may only have one output. For every input of a function there is a designated output; No matter how many times the function is run with that input, it will always result in the same output.

The Second is that functions do not access or manipulate variables or values outside of themselves. This means that functions are not able to work with global variables in their computation. Only the variables passed into the functions and variables defined within the function are able to be used in any computation within the function. Building on top of this, functions do not create side effects. This means functions do not alter variable values or states outside of its own scope. All functions are effectively stateless and perform the exact same every time they are called with the same input (as mentioned in the above point).

Lastly, all data in functional programs are immutable. When a variable is declared, its value is not able to be changed. This makes values much more safe to use in complex operations that may share the same values. Building on top of this, functional programming languages do not support loops that are found in imperative domains, such as for and $while$. Because values are immutable, the condition value in a $while$ loop would never change, likewise with the for loops, the variable defined in its scope would never be able to increase or decrease in value, making the loop impossible in the functional domain. Rather than using loops, however, recursion is used when looping is necessary in some computation.

Functional programming is a shift in how programs should be written. This shift makes for code that is more safe and easy to understand. In the following sections, this document will go into depth of the Artifice programming language and provide information on the data types and structures of the language as well as how the language is written, used, and understood.

Domain Specific Language

Artifice is a general purpose functional programming language built without a specific domain in mind. There are a number of general purpose programming languages that exist today, so while there are a number of languages to choose from in that regard, Artifice aims to be a hobby/playground/research language. The functional domain, while gaining traction in use, is still regarded as a fringe domain to build production level code with. There is plenty to be done in terms of functional language development, and Artifice is a testing ground for that.

Beyond functional programming, logic programming is another domain of programming development that has not been brought out of the academic sphere much. Artifice is the perfect language to begin to test aspects of logic programming being brought into a toy language to play around with and to understand its respective use cases better.

Getting Started Artifice is written in C and compiled with the *Clang* compiler and compiles down to llvm byte code. Apart from this, the lexer for Artifice is written in *Flex* (also known as fast lexical analysis generator), and the parser is written with GNU Bison. There is currently NOT a build tool available, so all library requirements should be installed independentally. Artifice Relies on Clang, Bison, and Flex. Artifice is hosted on github https://github.com/JustSomeCarbon/Artifice/tree/mainhere and can be cloned to a local machine using git. After Artifice is installed, it can be built by typing the following into the console while within the $Artifice/src$ project directory:

lstlisting *make*