# The Creation of a Programming Language

Florian Donnelly
Gymnasium Burgdorf

## Abstract

This is an abstract...

# The Creation of a Programming Language

## Contents

# Introduction

## Relevance of the Topic

## Goals

Listing 1: src/main.c

```c
#include <unistd.h>

#include "interpreter/interpreter.h"

int main(int argc, char** argv){
```

# Theory

## What this is and what this is not

The final product in the process of creating this programming language will not be perfect, or in fact, be used or adopted by anyone at all, and it is not intended to be. Creating a good, or even usable, programming language implementation by today's standards means to create an entire ecosystem of tools, not only including a compiler or interpreter, but also a package manager for dealing with dependencies and a platform for people to share their work.

The process used in this paper is more so a very core implementation of what a programming language, only containing the very necessary features.

# Implementation

## Used Software

At the heart of creating a program in C stands the C compiler. This project uses *gcc*, the GNU Compiler Collection, which, as the name suggests, also supports different languages than C. Substantiating the development process is *GNU Make*, a general-purpose build system. It handles tasks like building the executable step-by-step from the project source code and directory clean-up. The C ecosystem is very easy to use and fundamentally supported by *GNU+Linux*, the operating system used to program on. The main code editor used was *Microsoft Visual Studio Code*, providing great features like built-in git integration. *Git* is the project management and tracking software used to back-up and share the project files on *GitHub.com*, a Microsoft hosted git repository server to store and collaborate on source code.

# Results

# Discussion

## Interesting Ideas untouched

Creating the ideal programming language takes a lot of planning and sketching out. This section of the paper gives insight on interesting ideas or concepts that are used in other programming languages, not -the language- .

### *Foreign Function Interfaces*

A foreign function interface[1] often abbreviated as FFI, is a mechanism by which one programming language can call functions written in another. This makes a language more attractive, as it can be used with existing libraries from another language, instead of functions having to be re-written. Enabling C function calls often allows a certain language to directly interact with system libraries or the operating system itsself.

Foreign function interfaces are available in many modern languages. Examples are Java, Python and Rust. Java enables users to call C, C++ and even assembly code with the Java Native Interface[2], JNI for short, whereas within Python C function can be run with the standard ctypes[3] library.

Foreign function interfaces are specially difficult to implement, or implement right. Let us assume a programming language interpreter -the language- written in C that would like to call some functions that are part of yet another C program or library. The problem lies in the fact that at runtime, C does not know structure of the functions to be called, say return- or argument-types, which is by the way where the term foreign functions originates from. This is why the -the language- developer would have to provide that information by re-declaring the function inside -the language- , so that, with some trickery, data can be passed to and received from the function in the correct format, preventing a segmentation fault. The libffi[4] C-library, which is also used by Java and Python for this purpose, can be used to load and call such foreign functions.

### *Reflection*

Reflection[5], also called reflective programming, describes the ability of a program to examine and modify its own structure and behaviour at runtime. An example would be self-modifying code. This can easily be achieved in assembly language, which inherently does not differenciate between data and instructions. Another example is Java, providing methods and classes in the java.lang.reflect package that enable developers to examine and change properties or functions of objects. The advantage of reflection over not having it is that it allows for non-static programs and program flows, effectively making smaller executables and more resource efficient programs, as those programs can adjust themselfs to run differently under different conditions at runtime. The easiest and simplest way to go about implementing Reflection in -the language- is certainly to allow the interpretation of strings at runtime. That way, strings holding program instructions can be created and modified and later executed.

---

[1] Wikipedia, 2022a.

[2] Wikipedia, 2022b.

[3] Python, 2022.

[4] Github, 2022.

[5] Wikpedia, 2022.

## Index

## Closing Words

## Declaration of Autonomy

Hereby I, Florian Donnelly, declare to have written this paper, also including the entire source code of -the language- by myself using resources listed under the references section.

## Appendix

## References

Github. (2022, September 1). *Libffi*. https://github.com/libffi/libffi

Python. (2022, September 1). *A foreign function library for python*. https://docs.python.org/3/library/ctypes.html

Wikipedia. (2022a, September 1). *Foreign function interface*. https://en.wikipedia.org/wiki/Foreign_function_interface

Wikipedia. (2022b, September 1). *Java native interface*. https://en.wikipedia.org/wiki/Java_Native_Interface

Wikpedia. (2022, September 1). *Reflective programming*. https://en.wikipedia.org/wiki/Reflective_programming