Can I create an interpreted programming language?

My EPQ was to create an interpreted programming language capable of meeting the following success criteria:

* Can output data onto the screen
* Can store, manipulate, retain and output data in memory
* Can prompt the user for input
* Can make logical and arithmetical decisions and comparisons based on data stored in memory and data given by the user
* Can perform mathematical operations on both integers and floating-point decimals
* Must gracefully handle errors in both interpretation and at runtime

As well as the following which are not required but would provide additional functionality

* Basic networking in the form of HTTP GET and POST requests
* Basic file handling such as read, write, and read into memory
* Functions and a more Object Orientated approach
* Basic set of loops (for, while, foreach)

What is an interpreted programming language?

This question can be broken down into two main parts. What is a programming language? and What is an interpreter?

*What is a programming language?*

Firstly, a programming language is a series of pre-defines rules and grammar that can be used to produce a result of some kind by either an interpreter or a compiler. There are different levels of programming language, ranging from low to high level. The lower the level, the closer it is to computer readable instructions. The lowest level code is binary instructions which are decoded and executed directly by the processor. Everything higher level than binary has been designed to be easier to read, write and understand by humans.

*What is an interpreter?*

There are two different ways that programming languages can be converted from high level languages to low level languages and ultimately assembly languages and binary instructions, compiling and interpreting. Compiling is when a high-level language such as C++ is converted into an assembly language by a compiler. The advantages of compiling are that the program will often run faster than an interpreted program, but compilation can take a long time, depending on the size of the program and the speed of the programmer’s computer.

An interpreted language works by having an interpreter scan the program, executing each line as it gets to it. This is faster to run but large programs can be slower than compiled programs. DOG is an interpreted language as an interpreter is far simpler to write than a compiler and given the time constraints an interpreter is the logical choice, despite my interest in writing a compiled language.

Research

The majority of my research into this project was used to make an informed decision on whether to make an interpreted or compiled language, and the basic structure of an interpreter. This is because DOG is a large project and if the initial structure was wrong, there could be serious consequences in the latter stages of the project, as proved by my first attempt.

As programming is my hobby, I was drawn towards the complexity of creating a compiled language and the challenges it would bring, but after researching interpreters and compilers in more detail, I opted to go for an interpreted language, as there are significantly less components than a compiled language, and I since I was struggling in my other subjects I was unable to devote the time needed to write a compiler from scratch to EPQ.

My research involved looking at various documents, articles, papers and forums to discover what the best solution for my case would be. As with most specialisms, Computer Science and programming in particular can be biased, especially on forum-based websites, as the people working on similar projects can be very passionate about their way of doing things, so I had to be careful to look at things objectively in my research.

Insert some examples you used and issues around htem[[1]](#footnote-1)

I was very happy with my choice of an interpreter, as it enabled me to rapidly prototype and build the first versions of DOG and concentrate on adding new features to it to make it more useful rather than indulge myself with (lovingly) spending hour upon hour trying to make a compiler.

The other main thing I researched was specific details on my language of choice (C++), often while I was in the process of developing DOG. As programming and programming languages are very complicated, it is expected that developers need to research things while they are mid-project, often every few minutes, as there is simply too much to just memorise, especially if the developer uses multiple languages, which all have their own slightly different way of doing things. This kind of research often involved checking syntax and method names, and occasionally involved searching for simple algorithms or functions to perform a basic and mundane task, as re-inventing the wheel would be time consuming and probably result in a worse solution.

**Success of project**

Evaluate against your success criteria

Evaluate how useful in a general way your research was

**How to see the finished product:**

Give link

Refer to Appendix 1 the written code

Bibliography on separate sheet – list of all the sources you looked at and used

1. [↑](#footnote-ref-1)