**TINY**

Tokenised Interpreter for Non-complex Yield

-

1. Project Vision and Scope

*TINY* is a JIT interpreter coded in C++ for a custom basic arithmetic programming language. The language is designed with education and restricted challenge in mind. The intention behind the project is to be an entry point for people who want to learn basic programming, as well as experienced developers eager to experiment with more complex projects without the steep entry curve.

*TINY* is open-source – unlike many beginner-friendly languages – so you can easily see what the interpreter is doing under the hood. In addition, we provide official documentation describing how it works, which makes learning more approachable. Coding enthusiasts can even fork and modify the source code to tailor their needs.

We will never introduce high level functionality or deliver a versatile product meant for serious production use. *TINY* is only supposed to be a concrete foundation on which anyone is able to expand with little effort and time.

Scratch is an accessible tool for game dev. *TINY* is the equivalent for JIT interpretation. We are all about introducing a way to learn not only through engaging the language itself, but also by diving into the cogs turning within, cracking the books of code execution.

1. User Personas

* The curious of all ages with no prior experience with programming languages who want to learn numerical computing in a simple inviting environment
  + Users will want to use our product for its ease of use and readily available sources
* Coders on an intermediate skill level who want to familiarize themselves with JIT interpreter implementation and lower-level programming
  + Users will want to use our product because of its open-source nature
  + Users can build upon TINY by designing new features or experimenting with modifications to its syntax or core functionality
  + There’s no popular alternative for this user group
* Hobbyists, educators, and enthusiasts interested in language design.
  + This appeals to those interested in gaining practical experience with language engineering and compiler/interpreter design.

1. Use Cases
2. For diving into coding

|  |  |  |
| --- | --- | --- |
|  | Steps | Outcomes |
| 1. | Download the TINY.exe application (a pre-compiled interpreter ready for use) from our website | You’re all set up for coding in TINY. |
| 2. | (optional) Create a \*.tiny file to write your code in | This will allow you to write complex sequences of commands to be executed at once in series. |
| 3. | (when using an IDE to write code) Visit the IDE guide on our website for instructions on how to install essential plugins for your editor | Our plugins provide syntax highlighting, auto-completion, facilitated access to offline documentation and various project templates. |
| 4. | Follow the online documentation and write your commands | This will enhance your problem-solving capacity and make you ready for writing more complex code in other programming languages without overwhelm. |
| 5. | In case of encountering issues, refer to our online Get Started Guide | The Get Started Guide provides a simpler version of the text instructions from the documentation expanded for those less knowledgeable of computers. |

1. For advanced tinkering

|  |  |  |
| --- | --- | --- |
|  | Steps | Outcomes |
| 1. | Download the TINY source code or fork the GitHub repository | You’re all set up for tampering with the codebase and learning to navigate through the modules. |
| 2. | Read the documentation and build the interpreter binary | This will provide the foundation for modifying and testing the language as you add or change features. |
| 3. | Visit the official TINY Devlog YouTube channel | This will provide you with a more in-depth look into the structure and implementation with commentary. |
| 4. | Modify the source code, implement new features, or experiment with altering the existing modules | This will familiarize you with the hows and whys of interpreters and improve your skills and understanding. |
| 5. | Learn about parsing and math operation mechanisms work | This will soothe your curiosity about the potential implementation of the inner workings of parsing strings to arithmetic. |
| 6. | Get to know the Triggered Injection for New Yield API and learn how to hook into internal events | This allows you to add custom behaviours or modify existing ones in an intuitive but extremely flexible way. |
| 7. | (optional) Use the TINY.exe executable as a CLI to create a project with the Triggered Injection for New Yield API template | The template provides a basic structure, boilerplate code and some examples, which facilitate the modification process. |
| 8. | Use the CLI to compile all created projects and integrate them into the compiler by patching the TINY.exe executable | This tool provides an automatic method of applying your work. |

1. Experimenting with Language Design and Extension

|  |  |  |
| --- | --- | --- |
|  | Steps | Outcomes |
| 1. | Fork the TINY GitHub repository or download the source code | You're all set up for experimenting with the codebase and exploring language design possibilities. |
| 2. | Familiarise yourself with the documentation and build the interpreter | This will provide the foundation for modifying and testing the language as you add or change features. |
| 3. | Visit the official TINY Devlog YouTube channel | This will provide you with a more in-depth look into the structure and implementation with commentary. |
| 4. | Implement new syntax or tweak the language rules, experimenting with language behaviour | By modifying TINY's syntax or structure, you'll gain insight into language design principles and the foundational components of programming languages. |
| 5. | Get to know the LDE (Language Design Extension) API and learn how to create custom syntax | This API allows you to implement custom syntax patterns and integrate them into the language. |
| 6. | (optional) Use the TINY.exe executable as a CLI to create a project with the LDE API template | The template provides a basic structure, boilerplate code and some examples, which facilitate the design process. |
| 7. | Observe and analyse how changes impact parsing and interpretation mechanics | This step will deepen your understanding of parsing, tokenisation, and JIT interpretation, enhancing your grasp of low-level programming and interpreter design. |