1. **Scripture**

*And by them their children were taught to read and write, having a language which was pure and undefiled*. Moses 6:6

2. **Abstract**

The focus of this project is to design and implement an interpreter for a garbage collected, multi-paradigm, dynamically typed programming language called Tealang.

3. **Background**

i. *Definitions*

Dynamic Typed – The programming language would limit type checking to runtime.

Interpreter – The programming language is not compiled into machine code. Instead, the statements of the language are fed to an interpreter on a line by line basis.

ii. *Why this topic is of interest*

Programming languages are a way to communicate human intentions to a machine. However, a programming language is useless unless there is a way to translate that language into tasks that the computers can perform. This translation of a human literate programming language to machine instruction is done through a compiler. While compilers are an interesting topic by itself, the principles behind the design of a compiler have other applications including software complexity management, algorithm development, machine architecture design, text processing, and so on.

iii. *Prior work by others*

Various programming languages have been implemented over the years. “Tealang” takes its ideas from various languages including the syntax of Objective-C, the 'everything-is-an-object' approach of Python and Ruby languages, and the concise and simple representation of data structures in the Lua language runtime.

4. **Description**.

i. *Summary*

There are two aspects to this senior project. The first is to design a programming language and the second to develop an interpreter for it. The programming language, Tealang, takes its inspirations from various languages including Objective-C, Python, Ruby, and Lua. The idea is to increase readability by forcing verbose programming constructs. This concept, borrowed from Objective-C, is complemented with the scripting environment like that of the Python and Ruby programming languages.

ii. *What constitutes success?*

The progress of the project can be tracked with the completion of specified milestones. Completion of all but the optional milestones would constitute a success. Specifically, the completion of the project would be indicated by a working implementation of a lexer, parser, syntax analyzer, intermediate language generation, and a virtual machine. Along with the completion of the specific aspects of the interpreter, the final phase of the project would involve testing. The test cases would attempt 100% grammar coverage and the implementation of common programming tasks and algorithms. If the interpreter is successful in executing these test cases, the project is considered a success.

iii. *Tasks*

a. Preliminary Research and proposal preparation – The initial research would involve a general overview of the topics necessary for compiler design. It would serve as a starting point for further research into the topic.

b. Research – With the preliminary research providing a general knowledge behind compiler design, further research is done with an incremental approach. As a compiler can be divided into specific tasks, each task would be preceded by a research on that specific task.

c. Requirements and specification – The specification of the language would define the various constructs of the programming language.

d. Design – The design phase would include activities to consider the best approach to implement the language defined in the specification. This task would include consideration for the various data structures, algorithms, and architecture of the virtual machine.

e. Programming – The programming task includes the activities associated with implementing the design. This task is sub-divided into developing the lexer, parser, syntax analyzer, and the virtual machine.

g. Testing – The testing phase involves quality testing to make sure that the interpreter works for the language defined in the specification. It would involve using test cases to verify 100% coverage of the language grammar as well as to test the usefulness of programming language for common tasks.

5. **Scope**

The scope of the project is to deliver an interpreter that is able to read a Tealang source file, parse it, and execute the instructions.

6. **Schedule**

The hours are estimated on the assumption that one week would produce 30 hours of work. The programming phase is done in an evolutionary approach and, therefore, there is no specific start and stop date for the sub-level milestones.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Milestone | Start Date | End Date | Estimated Hours | |
| 1 | Preliminary Research and proposal preparation | March 10, 2014 | March 17, 2014 | 30 | |
| 2 | Requirements and Specification | March 17, 2014 | March 24, 2014 | 30 | |
| 3 | Design | March 24, 2014 | April 7, 2014 | 60 | |
| 4 | Programming | April 7, 2014 | July 1, 2014 | (300) | |
|  | i. Lexer |  | | | 45 |
| ii. Parser |  | | | 45 |
| iii. Syntax Analyzer |  | | | 60 |
| iv. Intermediate Language generation |  | | | 60 |
| v. Virtual Machine |  |  | 90 | |
| 5 | Testing | July 1, 2014 | July 17, 2014 | 60 | |
| Total | | | | | 480 |

7. **Deliverables**

The two deliverables for the project include the language specification and the source code.

8. **Applicability**

Building a compiler is a complex task. It is not possible to simply “hack code” to build a compiler. There is a need to specify the requirements, design the system, research, and then proceed towards building the system. That is, designing and implementing a compiler requires software engineering. Developing a compiler would allow me to work at a scale relatable to the real world.

Aside from requiring careful software engineering, compilers are also algorithmically intensive. For an interpreter, which is the scope of this project, speed becomes a major issue. Choosing good algorithms and data structures becomes a core requirement.

9. **Required Resources and Costs**

Most of the resources necessary for this project are available freely online, through the ACM and IEEE digital libraries or through the BYUI library. The two books that I am using as primary references are the following:

1. *Compilers: Principles, Techniques, and Tools – $131.48*

This book is the main source of reference for this project. It is borrowed from the library.

1. *The C++ Programming Language, 4th Edition - $55.75*

The book serves as a reference to the C++ language. I currently own the book.

10. **References**

1. Aho, Alfred V., Ravi Sethi, Jeffrey D. Ullman, and Monica S. Lam. *Compilers, Principles, Techniques, and Tools*. N.p.: Pearson Education, 2006. Print.
2. Ierusalimschy, Roberto, Luiz Henrique Figueiredo, and Waldemar Celes. *The Implementation of Lua 5.0.* Journal of Universal Computer Science 11, 2005. Web. 10 Mar. 2014.
3. Brian Davis, Andrew Beatty, Kevin Casey, David Gregg, and John Waldron. 2003. *The case for virtual register machines*. In Proceedings of the 2003 workshop on Interpreters, virtual machines and emulators (IVME '03). ACM, New York, NY, USA, 41-49. DOI=10.1145/858570.858575 http://doi.acm.org/10.1145/858570.858575