Scribble - Design and roadmap

This third year project will be about producing a small programming language and virtual machine in C++.

Scribble is a embeddable programming language designed to be used as a low cost solution (In terms of space and price and time) for anybody trying to extend the functionality or reduce the complexity of their application. It will allow a developer to interact with user generated content via a two way API, functions declared in Scribble can be executed from C++ and functions declared in C++ can be allowed to execute in Scribble. It has been designed to simplify a development process; It does this by allowing developers to offload less intensive or important code into a easier garbage collected language whilst maintaining a strict environment with clear types and promoting a clean codebase.

The Prototype

Work started on a prototype of the language to prove to myself ahead of time that I had the technical skills to achieve what I wanted after a dialogue with my supervisor toward the end of my second year.

This has led to me laying out a three part design which includes the parser which takes the Scribble code and converts it into a set of functions and types and generates intermediate code for the virtual machine, a intermediate Parser named SASM which generate the bytecode and a virtual machine which will run bytecode generated by the SASM parser.

This prototype has clearly highlighted the portions of the application that I am likely to struggle most with are ensuring a conflict free grammar, memory handling and code generation which should drastically improve my ability to budget time.

The prototype used a similar tree generation parser to the one proposed in this project however the execution of the language was done on the tree. The grammar also caused issues as Bison could not generate a parser without potential conflicts for it. However the basic syntax of the language and the structure of the tree is likely to remain which will allow me to reuse large portions of code in the construction of the new one.

I also prototyped a simple virtual machines similar to LUA's VM design however with significantly reduced functionality. This will allow me to save time when designing the new virtual machine by providing me with existing experience and a code base to draw from.

These prototypes should help me achieve my initial ambitious time scale. However neither of them included any kind of formalization which will be the focus of my efforts when developing the full project.

Technical background (Abstract)

Scribble

The language

The language will be split into at least three possibly four separate sections during development. The grammar which Bison & Flex will use to generate the parser and which will define how the language is written will be one section. The glue which handles the actual grabbing of the files and handing each of them over to bison will be another and finally a tree like structure that will be generated by Bison will handle both intermediate code generation and code verification (Checking that code that has a valid syntax isn't incorrect in some other way such as type abuse).

There will also be a large amount of testing and refining required during this period as it's hard to gauge how well an idea will work until it is put into practice. While this would be hard to budget time for due to its unpredictable nature it is likely to be a large portion of the time spent working on the language.

The language itself will be strict language where everything has a predefined type. Due to the ambitiousness of the project to save time as much of the grammar and tree structure that can be reused from the prototype will be. It will take ideas from languages such as GoLang and LUA and its syntax will resemble a C style syntax, with a few exceptions such as type being declared to the right of definition. My final aim deliverable for the language is to have a language with a clean syntax that lends itself to obvious understandable code whilst remaining concise and a parser that is quick (enough) and as bug free as possible with an emphasis on making sure there is no ambiguity in the syntax (No two rules that conflict or could cause issues with the language).

ScribbleASM

The intermediate language

The intermediate language is used as a stepping stone between the complex high level Scribble parser and the low level bytecode that the Virtual Machine accepts. It will take human readable pseudo-assembly generated by the higher level parser and converts it to bytecode; this step is mainly for debugging allowing me to have a look at a human readable version of the bytecode being generated.

The instructions will usually follow the same format as their bytecode equivalent (For example, the arguments will be in the same order in SASM that they would be expected to be encoded into the instruction).

ScribbleVM

The virtual machine

The design of the virtual machine demands both a formal specification and an implementation. Though they will probably be developed in parallel they should be entirely separate entities, the specification documenting the expected behaviour of every type of instruction while the virtual machine will contain my own implementation of the specification.

It will be a register based virtual machine with 64 bit instructions, this is to allow myself the largest window for expansion given that memory constraints and space constraints are unlikely to be a problem with any of the processors and systems available today.

A formal specification is required as without a specification the virtual machine would be too inconsistent and ambiguous and the project would break often. It will include a description of the bytecode that the virtual machine is capable of handling as well as a broader description of how things like memory should be handled. It should also be clear enough to allow any third party to implement their own virtual machine capable of executing the same bytecode.

The implementation on the other hand will be a lot more fluid than the specification. This is the first virtual machine I have attempted to implement from the ground up capable of doing anything non trivial (Having only looked at emulation of older devices in the past and prototyping over the summer) and thus I am bound to make plenty of mistakes or design choices which I later may decide to rip out or completely change. The project is also very large and having the formalized specification will help me deal with the scale better during implementation as I will be much more capable of identifying crucial and less crucial parts of the program.

Timeline

The work listed on this time line will continue on from the prototype developed during the summer. It is because of this I have decided to work on the virtual machine first. By building upon the prototype I can spend time implementing a virtual machine specification at the start of the project and avoid the issues that would arise when attempting to design and formalizing a good syntax and a functional virtual machine at the same time.

As each portion of this project is fairly reliant on the other portions (with no Scribble it will be difficult to construct a virtual machine as there will be no code to execute, with no VM Scribble will be unable to execute code etcetera) it is likely that often whilst working on a feature for one I will need to implement a feature for the other. This is another reason for tackling the lower level portions first however it may still be the case that often a feature for one leads to changes in all three portions of the program.

During the development I will be constructing and updating a more short term roadmap containing lists of planned short term features (such as implementing a new type) per iteration within each of the broader sections defined here. It would be difficult to write this now as more in depth plans are bound to change when work has begun.

A Gantt chart showing the timeline for the first term will be provided with this document.

Term One

Construction

Tools

Over the summer I have been collecting and building a set of tools to make the development process easier. These tools include scripts to handle automatic build number incrementing and the Makefile will need to be cleaned up and refined (In their current form they are littered with relics from failed experiments). Some time will need to be spent cleaning and adapting this tools so that I do not have to worry about them later. These tools will become an integral part of the project, most are likely to be executed every single time the project is built and if one of them is faulty it could lead to some really cryptic debugging sessions. **The work will be completed by the end of the first week of term.**

Virtual machine development

The virtual machine is one of three integral portions of the project. It will involve the development of a formal specification and an implementation.

Formal specification - The formal specification for the virtual machine will be written in conjunction with the development of my implementation of it. Initially most of the work done will be regarding formal definitions of each possible OpCode within the VM. Making sure that they provide the functionality required and also making sure that I don't have too many. Later the specification will include provisions on how the memory should be handled and on how types should be defined.

The formal specification will be completed by the first week of the second month of term however it will be updated as the requirements of the project change.

VM Implementation - My virtual machine will be designed using iterative model, initially supporting only base operations each next opcode being devised to suit a specific issue or need that I come across when trying to make it compatible with the requirements Scribble the high level language it has to support. I have decided to take this approach so as to gauge the real world usefulness and viability of proposed features of the formal specification.

The implementation will begin development will begin with the design of the virtual machine. After the initial virtual machine specification is formalized the virtual machine implementation will be aggressively developed and maintained to keep it up to date with the specification.

Intermediate language development

The intermediate will be comprised of a bison & flex, a set of interfaces allowing a developer to easily convert intermediate code to bytecode and a formal set of clear documentation explaining the syntax of the assembly language and the result of each possible statement. **Work will begin in the second week of the second month of term. It should take one week to complete.**

Language development

Time allocation on the language will be split down into several subsections.

Grammar - The grammar will likely be one of the first sections of the language finished. It will be used by Bison to construct a parser which will form the tree structure described in the design section. This is the area however that is most prone to issue (Bison Reduce/Reduce conflicts) and is also the most subjective (What I think is a good idea may be horrible in practice) **One month for an initial formal specification. From then on any new features will be formalized in the specification before being prototyped in the grammar and then the tree.**

Tree - The tree structure will check validity of each statement within the language and construct the intermediate code handed to SASM. It will be initially time consuming but once constructed is unlikely to require to many changes before the optimization stage (This is where unnecessary operations will be removed and potential optimizations will most easily be seen). **The tree will be developed in tandem with the bison grammar. The initial tree structure will be taken from the prototype and the necessary changes will be made to strip away all of the derivation tree based execution and prepare it for a virtual machine.**

Glue - The bindings and wrapper that will allow the unwieldy Bison interface to exist and look nice within a C++11 environment. These will not be very time consuming and most can be adapted from the prototype. **Work on the interface should take a week.**

Work will begin from the half way through second month. The estimates for each section will be highly unpredictable but it is likely that an initial version will be created within the first month.

Weekly iterations

From the middle of the third month work should have progressed to the point where the language and virtual machine portions the project demands a similar amount of work. From this point on I will be proceeding with weekly sprints each focusing on a feature or a set of features from either the virtual machine or the language. **The middle of the third month until the end of the first term. Potentially spilling over into the break if time constraints apply.**

Term Two

Interfacing & Refinement

For the second term I intend the focus of my work to be on optimization and bug fixing. Having a stable environment is integral for the development of a product and no sane developers would adopt a platform that was slow and crashed. Because of this need for stability most of the changes will not directly affect what a user sees but will instead be largely internal with a strong emphasis on improving what is already there rather than implementing new things. I also intend to develop a user friendly API allowing developers to easily interface native and Scribble code will be developed. However these objectives are low priority. allowing me some room to breath if the more difficult features tackled in the first term take more time than planned.

The API

Once the VM is properly defined and implemented work can begin on an equivalent to the JNI or other native interfaces which will allow two way communication between the a developer in C++ and the functions executed on the Scribble virtual machine. **Work will begin at the start of the second term. It should be completed in one month. A large amount of time is being left so that various methods of interfacing can be trailed.**

Feature freeze

From this point on bug fixing and speed will be the focus of development in order to get it into a polished state for the end of the project. New features will not be developed unless they are integral to the project. **This will come into force at the end of the first month of the second term.**

Polishing reports and formal specifications

From this point on OpCodes, memory management etcetera will be set in stone and so each of the specifications and my final report draft will need going over to make sure that they are correct and to find any areas of ambiguity. I want to ensure that any client gets exactly what is described in the specification and that none of the behaviour of my code is ambiguous or contradicts the specification. Whilst it is unlikely that this will be entirely true as there are always bugs the issues should be as small as possible. **This will be continuing from the third week of the second term until the end of the project.**

Optimization & Bugfinding

Scribble should be as difficult to crash as possible by the end of the project. To this end the remainder of the second term will be spent looking for issues which could cause the virtual machine to fail ungracefully (It should never segmentation fault and always freeze after alerting the user to an issue and hopefully producing a detailed lot) and issues in code generation causing the language to not work as intended. This is likely to be quite a lot of work as Scribble will have an extensive code base by the end of the project. This is also being left tell last as if more time for features is required this can be overlooked. **This will be continuing from the second month of the second term until the end of the project.**

Deliverables

The first deliverable will be a report documenting the design process of the virtual machine. This document will list how the virtual machine has been designed and what potential limitations there could be from the choices made up to that point. This will be produced by the end of the third week.

The second deliverable will be an initial specification for the virtual machine documenting it's inner workings and opcodes and other technical details. This will be produced by the end of the fourth week and will be accompanied by the virtual machine implementation. This is likely to only cover a minor subset of the final specification which will be designed iteratively later in parallel with the language.

The third deliverable will be the intermediate language parser and specification completed half way through the second month of the first term.

The fourth deliverable will be a document defining the language syntax, explaining how each feature of the language works and documenting why the I have made selected design choices. This will be produced by the end of the second month

A report will be produced at the end of the last month of term documenting the progress made in the sprints.

The next deliverable will be a set of API user documentation explaining all of the interfaces and classes as well as giving examples of how to incorporate Scribble into a project. This will be produced by the end of the second week of the second term.

The final software deliverable will be a package which allows the user to compile and execute Scribble code with as few technical issues as possible. This will be delivered at the end of the project along with the project report.

Testing

Through this project I will be using C++ testing tools to maintain a set of tests to verify the functionality of the virtual machine, intermediate language and parser. The virtual machine tests will be constructed to the test that the virtual machine adheres to the specification by testing the virtual machine state after executing specific sets of bytecode. The tests for the intermediate language and Scribble itself will build upon these verifying that a piece of high level code gets converted into the correct intermediate and bytecode and that the expected result of any execution is correct.

Reading

During the summer I spent some time reading both about different parser technologies, this lead me to use Bison for the project (Due to a wealth of online resources and extensive platform support).

I will be reading the dragon book (Compilers: Principles, Techniques, and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman) and also using the Elizabeth Scott's notes for the third year compilers course to aid in constructing a stable grammar and to bolster my knowledge in the more theoretical portions of the project. I will also be reading various technical documents relating to the LUA and other virtual machine based languages to investigate potential design choices.

1. Compilers: Principles, Techniques, and Tools by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman, 1986

2. Elizibeth Scotts notes on compilers and code generation