Independent Project Report
"Qu" Programming Language Design

Pham Lan Phuong
Fulbright University Vietnam

Fall 2023

1 Introduction

This document concludes and reports the results of an Independent Project to develop a simple programming language named *Qu*. Qu is a non-Turing-complete programming language designed to queue objects on a presentation slide. For example, if an user wants to display "image1" first, then the text "text1", they can express it as the following code:

```
<1-> image1
<2-> text1
```

which, after rendering to the presentation slide, would behave as expected.

2 Project Motivation

Currently, managing the order of object appearances on a presentation slide requires considerable time and multiple clicks. For Microsoft PowerPoint and Google Slides, a user must select each object they wish to modify, navigate to the "Animation" menu, scroll to locate the appropriate effect, and then select "Apply." For LaTeX Beamer, image controls are slightly more convenient, but a lot of copy-paste would need to be done in order to achieve custom order. Python Manim module has great results but it demands a lot from the user, since the code library has become more and more complicated to serve the needs of advanced users.

Qu aims to provide the great control that Latex Beamer and Python Manim has, without the extra copy-paste or a steep learning curve. The language is simple, and it serves to do one thing only: queue objects so that they appear in the desired order. This facilitates the presentation process of students and professors alike, making well-ordered presentations less time-consuming.

The programming language has two main parts: the language design and a compiler. The design of the language is determined by the context-free grammar that specifies the production rules.

The first part of the compiler takes in source code written in Qu, creates a matrix that shows the status (visible = 1, invisible = 0) of each object in each slide. This matrix clearly expresses all the objects and its visibility on each slide, making the process of automating slide making easier.

In terms of scholar development, this project presented an abundance of challenges for me to explore and work through. Most of the necessary theoretical knowledge I have learned from CS302 - Theory of Computing class at Fulbright. This project provides practical experiences in language design and compiler building, and I strongly believe that there is always a gap between theory and practice that can be filled by hands-on experience. These skills are highly relevant in modern software development, where designing domain-specific languages and creating compilers or interpreters are increasingly valuable.

Overall, this project would equip me with multiple technical insights and problem-solving abilities necessary for my future endeavors in software engineering.

3 Language Design

3.1 Alphabet

The alphabet of Qu is the same as Python's alphabet, which could be found here[1].

3.2 Production rules

The full production rules can be downloaded from this Github repository. It is rather long and thus will not be included in full here.

3.3 Language Parser

The language parser is written in Python, and can be found in the this file[2].

The parser for Qu takes a Qu source code file (or string) as input. It works by iterating over each character in the source code file, identifying delimiters such as `"<"` and `">"`, and tokenizing the code. The output of this program is a matrix of size nxm, where n is the number of objects in the entire presentation, and m is the number of frames in the presentation.

4 Project Deliverables

All can be found here: Qu Github repository. The deliverables of this project includes:

- Grammar specification: A comprehensive document outlining the alphabet, production rules, and sample code for the Qu programming language
- Parser source code: The implementation of the parser, whose input is Qu source code, and output is a matrix displaying object visibility for each frame

To keep the deliverables of this project realistic, the first version of the compiler should work with two main objects: images and textboxes (one textbox can have multiple lines of text). Other objects that might be incorporated to make this language more useful are (but not limited to) tables, mathematical formulas (LaTeX), code snippets, which is more realistic to reserve for future developers.

5 Project Learning Reflection

There were several key skills that I wanted to practice with this particular project. First, and most importantly, I aimed to gain practical experience in language design and compiler design. I had learned about the fundamentals of these areas in Theory of Computing, but until then, it had been a black box whose inner workings I did not understand. By creating Qu, I learned how to create a language that was simple, yet powerful enough to serve its specific purpose of queuing objects in a presentation slide. This required me to explore and consider language features, making choices and trade-offs between simplicity and convenience, and overall making informed decisions.

Another challenge that I tackled was resolving syntactic sugar in the language. Many languages have shorthand expressions to make writing code more succinct and convenient. Qu also had the shorthand notation to select multiple rows of the same textbox. For example, when writing “text1[0:3]”, a user wanted to target cells 0, 1, and 2 of object “text1”. The compiler was able to see through this syntactic sugar to understand which exact cells it needed to select.

The third tangible skill I improved was handling bugs and errors (debugging programs), as well as code documentation. All programs must have bugs, it is inevitable, and by undertaking a demanding project such as this compiler, I developed better practices to navigate my way out of the compounding error messages. When the first working version of this project was completed, I also learned how to write proper documentation so that future maintainers or developers would have an easier time understanding my code and my intentions.

Overall, throughout this project, I developed my critical thinking and problem-solving skills. As with many things, going from theory to practice had a large gap that was not easy to navigate. I found a lot of my theoretical understandings challenged, and thus I also did a lot of ad-hoc research in order to make sense of the obstacles I faced. I also experienced several "Aha!" moments, when something I did not know before started to make sense, which was honestly the part that excited me the most.

Acknowledgement

I would like to thank Dr. Huynh Viet Linh for instructing me throughout this project. I would also like to thank Tran Ngoc Anh for her support during the development process.

References

[1] *10. Full Grammar specification — docs.python.org.* `https://docs.python.org/3/reference/grammar.html`. [Accessed 21-05-2024].

[2] *GitHub - lanphgphm/Qu: A programming language used to queue objects in presentation slides making. — github.com.* `https://github.com/lanphgphm/Qu/tree/main`. [Accessed 21-05-2024].