Bilkent University

CS315 - Programming Languages Course Project

P.S!

Challenges Make Us Better

Nashiha Ahmed | 21402950

Mert Inan | 21402020

Cholpon Mambetova | 21402612

Instructor: Dr. Buğra Gedik

Section: 2

Date: 30th Dec 2016

Contents

1 | Introduction

The main ideology behind P.S! is describing a shape to a friend as if writing a letter. The name P.S! comes from the P.S. (Post Scriptum) in a letter, and the ! denotes a special meaning in our program, as explained in later sections. P.S! is a letter type programming language and will be compiled by a compiler named *Kukubo*, which means "drawing robot" in Japanese. The source code will look like an actual letter that people write and send to a friend giving instructions on some tasks. Therefore, the language is as human readable as possible, which is what makes it different from most other programming languages. Since P.S! is a figure-drawing programming language the main purpose of it is to draw shapes, from basic built-in ones to complex ones defined by a user. The programming language is made in such way that a user needs to be polite for the program to work. There are some reserved words like "please", "thank you" or "dear", that make sure that a user is polite in that sense. Therefore, we can say that authors of P.S! are propagating humanness and politeness. The idea of creating a letter type programming language came to us, while we were brainstorming for this CS315 project. While brainstorming we started to imagine that we were giving instructions to a friend. This is when we had our "eureka!" moment. This report deals with the challenges we faced and how we addressed them.

2 | Challenges

Start with the “Start Button” Challenge

Because we believe that facing and discussing challenges are positive and crucial to improvement, we want to discuss some of the challenges we faced in the second part of our project. Learning about BNF grammar in class and writing this grammar as exercises in class and during the quizzes suddenly seemed easier as we started thinking about writing a BNF grammar for our language. We simply did not know where to start. To tackle this problem, we searched existing BNF grammars for languages commonly used, such as Java. After examining the Java BNF grammar and parser, we broke down our language into its basic sections, which are the beginning and end of the entire program, the P.S. sections, the P.P.S sections, and the main.

A Parser’s Nightmare: Left Recursion and Common Prefixes

Left Recursion and common prefixes, oh my! We ran into these common problems in BNF grammar writing. We used the two techniques we learned in class: eliminating left recursion and left factoring.

The ∞ Shift/Reduce Problem

Another problem that we faced during the development was resolving shift/reduce conflicts. As most of the methods call themselves recursively, sometimes Yacc would unnecessarily invoke the conflicts. Hence, even though the conflicts exist, as there was no evident solution for creating the grammar perfectly unambiguous without knowing the associativity of all of the tokens, the shift reduce conflicts were left to the parser to be handled by its default action, which is shifting.

The Why? Why? Error (yyerror) Problem

In order to continue parsing after finding an error, Yacc’s error and yyerror functionalities were to be used. However, learning about them was, at first a challenge, as there were not many resources that our group could refer to, which explained the error concepts simply. Furthermore, as error handling is already done by Yacc, it was a challenge to change the default yyerror function to output the required string instead of the default error string. However, we handled this challenge, by researching more and finding the Bison documentation that explained yyerror usage in detail.

Less is More? More is Less?

We had originally written large incorrect sample programs for our parser. However, even though we implemented error checking after having checked one error, one error in a large program triggered other errors. For example, if a block did not include “Thank you.” at the end, the parser tried to parse the rest of the program which turned out to have errors as well because “Thank you” was not used to end a block. Therefore, we had to write new small sized sample programs.