HIGH POINT UNIVERSITY
Department of Computer Science

CSC4510 Programming Language Design and Translation

Project Details

Introduction

This project is worth 48% of the grade in this course. The project is to be done individually. You are strongly encouraged to start early on this project and schedule time each week to work on it. We will cover language translation/compiler construction techniques in the first half of the course but you will not be able to complete some coding aspects until after the corresponding class discussion. However, you are able to start early and the complete project is not due until the end of the semester. Note however, there will be other work assigned in the second half of the semester so you are strongly encouraged to complete the work early. Time management will be important throughout this project. The deadlines for the project were selected to provide you with flexibility and to allow you to manage your time and assignments in other classes.

The Project

You are to construct a compiler for the programming language "*lille*" (lille is the Danish word for small). Your compiler is to be written in C++ and a "starter kit" is available on blackboard. The compiler you write targets a Psuedo Assembly Language called PAL and the starter kit includes a complete interpreter for PAL code as well as a number of sample PAL code files that you can execute. Your solution must use the recursive descent technique described in lectures for the parsing but you can feel free to extend the functionality of the compiler in terms of its error recovery and code generation strategies.

To help keep you on track, there is a schedule of milestones which you must achieve, set out in a table below. Please enter the deadlines into your calendar and budget ample time to complete the work.

The code provided to you as a starter kit includes about 2,000 lines of C++ for the PAL machine and about 2,000 lines of C++ code for the compiler outline and the majority of the scanner. As a guide, my complete solution to the compiler is about 6,000 lines of C++. The code is downloadable from blackboard as well as accessible on `spock.highpoint.edu` at `~moudshoorn/CSC4510`. Feel free to download it and modify any aspect f the code you feel is necessary.

The Compiler Overview

Compilers are an essential part of the programming process. They translate source code from a human readable form such as C++ to a machine readable form (binary). In this project we will translate programs from the simple programming language *lille* to PAL, a simplified assembly language. This translation process includes the following steps:

Activity	Description
Scanning	Take the source code written in *lille*, and break it into tokens representing the components of the language.
Parser	Accept the tokens from the scanner and ensure that the program satisfies the syntax rules of the programming language. That is to say that the source code has the correct structure (e.g., declarations appear before statement, statements
Static semantic analysis	Check that the program satisfies the static semantic rules of the language (e.g., you don't try to add a number to a string, you call a function with the correct number of types of arguments).
Error recovery	Ensure that the compiler continues to process tokens while not generating excessive numbers of errors, or errors unrelated to a problem in the source file.
Code generation	*We will NOT be generating code for the target machine – the PAL machine as part of the project. However, if a student is interested in tackling this challenging phase, they are welcome to do so for extra credit.*

You will construct a compiler by implementing each of these steps in turn (with the exception of code generation).

Why not just implement everything at once? We need to manage the complexity of the task at hand, and debug the code as we develop it. In other words, we need to follow good software engineering habits in order to manage the task. You have about 4,000 lines of code to write and it is important to implement the compiler in stages in order to avoid being overwhelmed with debugging and resolving errors.

The Project Details

The writing of a compiler is a non-trival programming task. It is tedious at times, error-prone and complex. However, it all begins by understanding the task at hand and understanding the programming language that you will be translating and appreciating the target language and being comfortable with its behavior.

Before you write any code there are 2 tasks that you need to complete:

1. Read the handouts provided om the project, the *lille* programming language syntax and semantics, and any other documentation associated with each language construct.
2. Download the compiler starter kit. Read the code and experiment with it. Sample PAL code files are provided so you can see how the PAL machine behaves. You have the source code so you can see how the PAL machine is implemented. The code acts as the definition if anything in the PAL Machine handout requires additional explanation.

3. Read the C++ code which serves as a skeleton for your compiler and the basis for the scanner that you need to write (most of the scanner is provided and you only need to complete the segments related to strings and numbers. Pragmas are handled in the scanner, but you will implement them when you implement the static semantic analysis phase. The code you are provided also includes the majority of the compiler driver (you'll add code to is when you implement the parser etc), an error handler (all provided except for code to generate a listing file), an identifier or symbol table (skeleton code only – you'll need to develop this during static semantic analysis), a file to deal with exceptions raised within the compiler (complete), a symbol (complete) and a token (complete) class which are used throughout the compiler project. A makefile is also provided to help you compile the system. You'll modify the makefile as you create new classes etc.
4. Once you have completed the 3 steps above, you will be able to start on completing the scanner.
5. Once the scanner is completed and tested, you can commence work on the next phase.

Note that a number of sample *lille* programs and the corresponding PAL code has been provided. This is not intended to be a complete test set and you are expected to develop additional test programs to validate your compiler. The text cases used for grading will not be made available before the submission deadlines. It is part of your responsibility to develop test programs to ensure your code is correct.

lille is not intended to be a perfect language, nor is PAL intended to be perfect target assembly language. *lille* is intended to be representative of some language features and to give you an idea about how to implement them. As you develop test programs, and implement the compiler, think about the programming language *lille* and its language features, as well as way the language was defined and specified. These perspectives and your opinion will be helpful in the second half of the course as we examine programming language paradigms.

Grading of the Project

At each stage, you will be given a grade for that stage. The weighting for each of the stages is shown in the table at the end of this handout. All stages are due by 11:59pm on the date indicated in the table. Be sure that you plan your time in order to meet all of the deadlines – there is no problem with completing a stage BEFORE the due date and moving on to the next stage. Instructions on how to hand up each stage will be posted on blackboard.

You are expected to make a satisfactory attempt at all phases of the project. The exception is the code generate phase with is optional but carries bonus points toward your final grade. Partial credit is also available for serious attempts at the optional code generation phase.

The Schedule

Each stage is due by 11:59pm on the day indicated. It is expected that you plan for busy times (and any other unforeseen problems which might arise). Also note that nothing stops you from moving on to the next phase of your compiler prior to the due date – just take a snapshot of the code you want to submit and continue your work.

Grading for each phase of the project will focus of that phase only. That is to say, the scanner will not be graded during the parser phase etc. The final phase identified in the table below is an assessment of the compiler as a whole and examines all phases of the compiler and how well your compiler is able to translate the *lille* source code into correct, functioning PAL code.

Stage	Due Date	Value
Scanner	October 27	20%
Parser	November 12	30%
Semantic Analysis	December 1	30%
Error Recovery and Completed Compiler	December 10	20%
TOTAL		**100%**

Recall that this project represents 48% of the grade in this course.

Michael Oudshoorn
August 13, 2021