CS 4240 Compilers and Interpreters - Spring 2015
Project Phase 1: Front-end: Scanner and Parser and ASTs
Due: Thursday, Feb 19, 2015, 11:55 pm (via T-Square)

Introduction

This semester, through a project split into 3 phases, we are going to build a full compiler for a small language called Tiger targeted towards the MIPS machine. The phases of the project will be:

- Phase 1 (this phase): To build a scanner and parser along with AST generation for Tiger language.
- Phase 2: Semantic Analysis and intermediate representation (IR) code generation
- Phase 3: Instruction selection, register allocation and MIPS code generation

The purpose of this phase is to build a scanner and parser for the Tiger language using ANTLR version 3 (yes, we are going to use ver 3 since it supports abstract syntax tree generation which is what we need – not parse trees). First download and install version 3.5.2 from the URL: `http://www.antlr3.org/download.html`. You just need the binary that generates the Java target (from the input lexical and grammar specification of Tiger)—also download a rich set of examples, study the tutorials and documentation to get familiarized (we do not envision anyone making changes to the ANTLR source files).

Please use the language specifications for Tiger given in the reference document (separate from this one). It is a small language with properties that you are familiar with: functions, arrays, integer and fixed point types, control flow, etc. – the syntax and semantics of Tiger's language constructs are described in a separate document along with a small sample program.

Phase 1 Description

Part 1: Scanner

First, you will build a scanner that will scan the input file containing the Tiger program, and perform lexical actions and return tokens one-by-one on demand to the parser. The scanner will be built using the ANTLR version 3 above. First install and read the documentation for this tool and then use the existing examples to learn about the input format the tool expects and the lexer/parser it generates from the input lexical specifications and grammar specifications respectively. You will then describe Tiger's lexical specification to ANTLR in the format it expects to generate the scanner.

The scanner's behavior that checks conformance to Tiger's lexical requirements should be graceful; it should be able to read in the program's stream of characters and return the correct token(s) on each request. For lexically malformed Tiger programs, the scanner should throw an error which prints: the line number in the file, the partial prefix of the error (from the input file), and the malformed token in quotes with the character that caused the error pinpointed. The scanner should be capable of catching multiple errors in one pass – i.e., it should not quit but continue on after catching the first error. Test the token generation and error reporting on sample test programs – **each team is supposed**

to contribute a large Tiger program that exercises all the Tiger grammar specs. You are allowed to use/tweak ANTLR's error handling for lexing and parsing in our project.

Part 2: Parser and AST Generation

Generate an LL(1) parser for Tiger. This consists of three parts:

1. Rewrite the grammar given in the Tiger language specification to remove ambiguity by enforcing operator precedences and left associativity.
2. Modify the grammar obtained in step 1 to support LL(1) parsing. This could include removing left recursion and performing left factoring on the grammar obtained in step 1 above. It may even require the grammar to be refactored and re-written to remove some LL(1) show-stoppers. This is the main part of this phase. These two parts are to be done by hand.
3. The parser should then be generated using ANTLR.
4. Test the parser on a large test case of a Tiger program (each team is expected to contribute a large test case).

For syntactically correct Tiger programs, the parser should output "successful parse" to the standard output and also generate the parse tree as well as the abstract syntax tree (AST) that corresponds to the program. There are several ways to generate Parse Trees and ASTs in ANTLR—please first study the documentation and examples and then implement this part. Some third party projects also allow you to visualize the ASTs. We strongly encourage you to use them and see the generated ASTs, which will aid you greatly in debugging. During phase 2, we are going to walk or traverse those ASTs to generate the intermediate form so getting a thorough understanding of how ASTs look and can be walked, and visualizing them in this phase will help you a lot for the next one. For generating these, please use the format used by ANTLR.

For programs with syntactic problems, the parser is responsible for raising its own errors. In these cases, the output should be some reasonable message about the error, which should include: the input file line number where it occurred, a partial sentence which is a prefix of the error, the erroneous token and perhaps what the parser was expecting there. In addition, your parser should also output the sequence of token types it sees, as it receives them from the scanner when you turn on a debug flag in your code. This will help us in verifying your solution. For example, given the stream "`var x := 1 + 1`", the parser would output "`VAR ID ASSIGN INTLIT PLUS INTLIT`". In case of errors, the parser should continue recovering from syntax errors and pick up the rest of the errors in the program, reporting them all.

Part 3: Turn-in

Correctness

You can first test your front end using some simple Tiger programs. You are also required to contribute and test your phase on at least one very large Tiger program . You will turn in this program and the corresponding Parse trees and ASTs.

You may consult with the TA if you face problems with the tools. Please post your questions on Piazza and also help other teams out if you have answers to their questions. Each team must do its own work but discussions for resolving problems and help on general questions are fine.

Grading

Deliverables for phase 1:

1. Lexical spec for Tiger written in the ANTLR format. (10 points)

2. Generated scanner code. (10 points)
3. Tiger grammar in appropriate LL(1) grammar form in the ANTLR format. (30 points)
4. Generated Parser code. (10 points)
5. AST and Parse Tree Generation—modified parser. (25 points)
6. Sample Tiger program; its AST and Parse Tree; Testing and output report. (15 points)