Project 2: A Parser

Your combined project for this semester is to implement a compiler from a pedagogical language, Tiger, to the MIPS instruction set. The entire compiler will be implemented as a sequence of phases:

1. A scanner.
2. A parser.
3. Semantic analyses.
4. A generator for intermediate representation.
5. A MIPS code generator, which will perform instruction selection and register allocation.

The second project is to implement the parser.

1 Design of a Parser

The parsing phase of your compiler should interact with the scanner to construct the syntax tree of an input program, determined by the grammar given in the language reference. Given Tiger program P, (1) if P is a syntactically valid Tiger program, then the parser should generate the parse tree of P (defined in §2); (2) if P is not a syntactically valid Tiger program, then the parser should output a suitable error message (defined in §2).

You have a large degree of freedom in designing the parser, as long as it generates the correct parse tree of an input program. One reasonable design of the parser would be the following:

1. From the Tiger grammar G given in the language specification, generate an equivalent $\mathsf{LL}(1)$ grammar G' by removing left recursion and left-factoring the result. G' can be generated by a combination of manual and automatic work.
2. Generate the $\mathsf{LL}(1)$ parsing table of G'. The table can be generated by a combination of manual and automatic work.
3. Run the $\mathsf{LL}(1)$ parsing algorithm using the parsing table generated to construct a syntax tree T' for the input program P.
4. Transform T' to a parse tree of P under G. These transformations will remove parse rules introduced in the process of removing left recursion and left-factoring.

A second reasonable design is to implement the $\mathsf{LR}(1)$ parsing algorithm for the given Tiger grammar.

2 Evaluation

Please place all project files within a directory named `p2` within your team's `git` repository. Include a Makefile such that running the command `make` within `p1` generates an executable file `compile` that runs your compiler.

If the parser is given flag `--ast` and is given a valid Tiger program P, then it should output the *S-expression* of P to standard output. The S-expression of a syntax tree T, denoted $\mathsf{sexpr}(T)$, is a string representation of T, defined as

follows. **(1)** If T is constructed from only a terminal `T`, then the S-expression of T is `( T )`. **(2)** For each non-terminal A and grammar symbols $a_0, \ldots, a_n$, if T is constructed by applying parse rule $\mathrm{A} \to a_0 a_1 \ldots a_n$ to syntax trees $T_0, T_1, \ldots, T_n$, then $\mathsf{sexpr}(T) =$ `(` A $\mathsf{sexpr}(T_0)$ $\mathsf{sexpr}(T_1)$ $\ldots$ $\mathsf{sexpr}(T_n)$ `)`. If the parser is given both flags `--tokens` and `--ast`, then it should output the string of tokens, followed by the S-expression of the parse tree.

Independent of whether or not your compiler is run with flag `--ast`, if it is given an input that is not a syntactically valid Tiger program, then it should output a suitable error message to standard error. We will only evaluate this phase of the compiler on strings that represent valid sequences of Tiger tokens. If it matches the entire input as a sequence of tokens but could not parse the input, then it should output to standard error the string representation of the token that it read to determine that it could not parse the input.

We will run your parser on a program (stored in filename, say, `program.tgr`) by running the command

```
compiler program.tgr
```

(with optional flag `--ast`)

You have complete freedom in designing how your compiler is built, so long as the command `make` generates a suitable executable file `compiler`. In particular, you can use another build system to perform all of the actual build work and include a Makefile in which the target `all` simply runs the actual build command.

To evaluate your parser, we will run it on a set of test Tiger programs and compare the result to a reference answer. To partially automate grading, we will by default evaluate the compiler as correct on an input program only if the result is identical to the reference answer, ignoring whitespace. Given that the Tiger grammar is unambiguous and the S-expression of a syntax tree is determined by the tree, for any given Tiger program, there should be a single correct output. However, we will inspect each output automatically flagged as incorrect to confirm that the output is actually incorrect and to award partial credit on a case-by-case basis.

We may, at a future date, provide a reference implementation of a parser and a selection of test programs in the course repository.

3 Discussion

We will award identical grades to each member of a given project team, unless members of the team directly register a formal complaint. We assume that the work submitted by each team is their work solely. Any non-obvious discussion or questions about design and implementation should be either posted on the course's Piazza message boards privately for the instructors or presented in person during office hours or after lecture. If the instructors determine that parts of the discussion are appropriate for the entire class, then they will forward selections. Under no condition is it acceptable to use code written by another team. The instructors will automatically check each submitted solution against other solutions submitted and against other known implementations.