Assignment 2 Document

Work to submit

- Report
  - [X] LL(1) transformed grammar
  - [X] Remove all EBNF repetition and optionality constructs (grammar tool)
  - [X] Replace all left recursion by right recursion (grammar tool)
  - [X] Remove all ambiguities (ucalgary tool + by hand)
  - [X] First & Follow sets (grammar tool)
  - [X] Design
  - [X] Use of tools
- Implementation
  - [X] Parser (recursive descent or table driven)
  - [X] Derivation output
  - [X] AST output
  - [X] Error reporting
  - [X] Error recovery
  - [X] Test cases
  - [X] Driver

Steps to transform grammar to LL(1)

```
- Run it through the grammar tool to:
    - Remove optionality constructs
    - Remove repetitions (0 or more and 1 or more)
    - Remove left recursion
    - Generate UCalgary version for further processing
- Remove ambiguities:
    - First set clashes & Factorizations
```

Parsing Table (generated using the ucalgary website)

	plus	minus	or	lsqbr	intnum	rsqbr	equal	class	id
START								START → PROG	
ADDOP	ADDOP → plus	ADDOP → minus	ADDOP → or						
ARITHEXPR	ARITHEXPR → TERM RIGHTRECARITHEXPR	ARITHEXPR → TERM RIGHTRECARITHEXPR							AR RIG
ARRAYSIZE				ARRAYSIZE → lsqbr ARRAYSIZEAMB1					
ARRAYSIZEAMB1					ARRAYSIZEAMB1 → intnum rsqbr	ARRAYSIZEAMB1 → rsqbr			
ASSIGNOP							ASSIGNOP → equal		
CLASSDECL								CLASSDECL → class id OPTCLASSDECL lcurbr REPTCLASSDECL rcurbr semi	
EXPR	EXPR → ARITHEXPR EXPRAMB1	EXPR → ARITHEXPR EXPRAMB1							EX EX
EXPRAMB1						EXPRAMB1 → ε			
FACTOR	FACTOR → SIGN FACTOR	FACTOR → SIGN FACTOR							FA FA
FACTORAMB1	FACTORAMB1 → REPTVARIABLE FACTORAMB2	FACTORAMB1 → REPTVARIABLE FACTORAMB2	FACTORAMB1 → REPTVARIABLE FACTORAMB2	FACTORAMB1 → REPTVARIABLE FACTORAMB2		FACTORAMB1 → REPTVARIABLE FACTORAMB2			
FACTORAMB2	FACTORAMB2 → ε	FACTORAMB2 → ε	FACTORAMB2 → ε			FACTORAMB2 → ε			
FUNCBODY									
FUNCDECL									
FUNCDECLAMB1									FU TY
FUNCDEF									
FUNCHEAD									
FUNCHEADAMB1									
FUNCHEADAMB2									FU TY

	plus	minus	or	lsqbr	intnum	rsqbr	equal	class	id
FUNCPARAMS									FU TY RE RE
INDICE				INDICE → lsqbr ARITHEXPR rsqbr					
MEMBERDECL									ME VA
MULTOP									
OPTCLASSDECL									
OPTFUNCBODY									OP
PARAMS	PARAMS → EXPR REPTPARAMS	PARAMS → EXPR REPTPARAMS							PA RE
PROG								PROG → REPTPROG0 REPTPROG1 main FUNCBODY	
RELEXPR	RELEXPR → ARITHEXPR RELOP ARITHEXPR	RELEXPR → ARITHEXPR RELOP ARITHEXPR							RE AR AR
RELOP									
REPTCLASSDECL									RE VIS ME RE
REPTFUNCBODY									RE ST RE
REPTFUNCPARAMS0				REPTFUNCPARAMS0 → ARRAYSIZE REPTFUNCPARAMS0					
REPTFUNCPARAMS1									
REPTFUNCPARAMSTAIL				REPTFUNCPARAMSTAIL → ARRAYSIZE REPTFUNCPARAMSTAIL					
REPTOPTCLASSDECL									
REPTOPTFUNCBODY									RE → \ RE
REPTPARAMS									
REPTPROG0								REPTPROG0 → CLASSDECL REPTPROG0	
REPTPROG1									
REPTSTATBLOCK									RE ST RE
REPTVARDECL				REPTVARDECL → ARRAYSIZE REPTVARDECL					
REPTVARIABLE	REPTVARIABLE → ε	REPTVARIABLE → ε	REPTVARIABLE → ε	REPTVARIABLE → INDICE REPTVARIABLE		REPTVARIABLE → ε	REPTVARIABLE → ε		
RIGHTRECARITHEXPR	RIGHTRECARITHEXPR → ADDOP TERM RIGHTRECARITHEXPR	RIGHTRECARITHEXPR → ADDOP TERM RIGHTRECARITHEXPR	RIGHTRECARITHEXPR → ADDOP TERM RIGHTRECARITHEXPR			RIGHTRECARITHEXPR → ε			
RIGHTRECTERM	RIGHTRECTERM → ε	RIGHTRECTERM → ε	RIGHTRECTERM → ε			RIGHTRECTERM → ε			
SIGN	SIGN → plus	SIGN → minus							
STATBLOCK									ST ST
STATEMENT									ST ST

	plus	minus	or	lsqbr	intnum	rsqbr	equal	class	id
STATEMENTAMB1				STATEMENTAMB1 → INDICE REPTVARIABLE STATEMENTAMB2			STATEMENTAMB1 → ASSIGNOP EXPR semi		
STATEMENTAMB2							STATEMENTAMB2 → ASSIGNOP EXPR semi		
STATEMENTAMB3									
TERM	TERM → FACTOR RIGHTRECTERM	TERM → FACTOR RIGHTRECTERM							TE RI
TYPE									TY
VARDECL									VA RE
VARIABLE									VA VA
VARIABLEAMB1				VARIABLEAMB1 → REPTVARIABLE dot id VARIABLEAMB1					
VISIBILITY									VIS

First & Follow Sets (generated using UCalgary grammar tool)

Symbol	Nullable?	Endable?	First set	Follow set
START		Endable	class, func, main	$
ADDOP			plus, minus, or	plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm
ARITHEXPR			plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq, comma
ARRAYSIZE			lsqbr	lsqbr, semi, rpar, comma
ARRAYSIZEAMB1			intnum, rsqbr	lsqbr, semi, rpar, comma
ASSIGNOP			equal	plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm
CLASSDECL			class	class, func, main
EXPR			plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	rsqbr, semi, rpar, colon, comma
EXPRAMB1	Nullable		eq, neq, lt, gt, leq, geq	rsqbr, semi, rpar, colon, comma
FACTOR			plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	plus, minus, or, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq, comma
FACTORAMB1	Nullable		lsqbr, lpar, dot	plus, minus, or, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq, comma
FACTORAMB2	Nullable		dot	plus, minus, or, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq, comma
FUNCBODY		Endable	lcurbr	func, main, $
FUNCDECL			func	id, rcurbr, func, integer, float, string, public, private
FUNCDECLAMB1			id, void, integer, float, string	id, rcurbr, func, integer, float, string, public, private
FUNCDEF			func	func, main
FUNCHEAD			func	lcurbr
FUNCHEADAMB1			lpar, sr	lcurbr
FUNCHEADAMB2			id, void, integer, float, string	lcurbr
FUNCPARAMS	Nullable		id, integer, float, string	rpar
INDICE			lsqbr	plus, minus, or, lsqbr, rsqbr, equal, semi, rpar, colon, dot, mult, div, and, eq, neq, lt, gt, leq, geq, comma
MEMBERDECL			id, func, integer, float, string	id, rcurbr, func, integer, float, string, public, private
MULTOP			mult, div, and	plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm
OPTCLASSDECL	Nullable		inherits	lcurbr
OPTFUNCBODY	Nullable		var	id, rcurbr, if, while, read, write, return, break, continue
PARAMS	Nullable		plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	rpar
PROG		Endable	class, func, main	$
RELEXPR			plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	rpar
RELOP			eq, neq, lt, gt, leq, geq	plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm
REPTCLASSDECL	Nullable		id, func, integer, float, string, public, private	rcurbr
REPTFUNCBODY	Nullable		id, if, while, read, write, return, break, continue	rcurbr
REPTFUNCPARAMS0	Nullable		lsqbr	rpar, comma
REPTFUNCPARAMS1	Nullable		comma	rpar
REPTFUNCPARAMSTAIL	Nullable		lsqbr	rpar, comma
REPTOPTCLASSDECL	Nullable		comma	lcurbr

Symbol	Nullable?	Endable?	First set	Follow set
REPTOPTFUNCBODY	Nullable		id, integer, float, string	rcurbr
REPTPARAMS	Nullable		comma	rpar
REPTPROG0	Nullable		class	func, main
REPTPROG1	Nullable		func	main
REPTSTATBLOCK	Nullable		id, if, while, read, write, return, break, continue	rcurbr
REPTVARDECL	Nullable		lsqbr	semi
REPTVARIABLE	Nullable		lsqbr	plus, minus, or, rsqbr, equal, semi, rpar, colon, dot, mult, div, and, eq, neq, lt, gt, leq, geq, comma
RIGHTRECARITHEXPR	Nullable		plus, minus, or	rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq, comma
RIGHTRECTERM	Nullable		mult, div, and	plus, minus, or, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq, comma
SIGN			plus, minus	plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm
STATBLOCK	Nullable		id, lcurbr, if, while, read, write, return, break, continue	semi, else
STATEMENT			id, if, while, read, write, return, break, continue	id, rcurbr, semi, if, else, while, read, write, return, break, continue
STATEMENTAMB1			lsqbr, equal, lpar, dot	id, rcurbr, semi, if, else, while, read, write, return, break, continue
STATEMENTAMB2			equal, dot	id, rcurbr, semi, if, else, while, read, write, return, break, continue
STATEMENTAMB3			semi, dot	id, rcurbr, semi, if, else, while, read, write, return, break, continue
TERM			plus, minus, id, intlit, floatlit, stringlit, lpar, not, qm	plus, minus, or, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq, comma
TYPE			id, integer, float, string	id, lcurbr, semi
VARDECL			id, integer, float, string	id, rcurbr, func, integer, float, string, public, private
VARIABLE			id	rpar
VARIABLEAMB1	Nullable		lsqbr, lpar, dot	rpar
VISIBILITY	Nullable		public, private	id, func, integer, float, string

Design

For the parsing algorithm, I have implemented the Table-Driven parser.

Grammar

In the `grammar.rs` file, I defined grammar concepts to represent the grammar we are trying to parsed. The grammar is composed of `NonTerminal`, `Terminal` and `SemanticAction` symbols. A `TerminalSymbol` can be/holds any of the `TokenTypes` (defined in assignment 1 in `token.rs`). A `NonTerminal` can be any of the `NamedSymbols` (for example: `<prog>` -> `Prog`) found in the grammar. I have a `GrammarRule` struct that represents a rule defined in our grammar. It simply holds the left hand side `GrammarSymbol` and a list of `GrammarSymbols` for the right hand side of the rule. I also have defined `DerivationTable` which holds a list of `DerivationRecords` to keep track of the progress of the parsing derivation. Each `DerivationRecord` holds the current state of the parsing stack, the current lookahead token and the derived rule.

In the `data.rs` file, I have defined the first & follow sets for all the `NonTerminal NamedSymbols`. I have also defined a `HashMap` that represents the parsing table for the grammar. Given a `NonTerminal` and a `Token`, the HashMap can return a `GrammarRule` or nothing based on the parsing table. I could have written the algorithms to generate the first, follow and parsing table on the fly, but I decided that my energy was well spent elsewhere.

Parsing & AST Generation

In the `parse.rs` file is contained the table driven parsing algorithm implemented by the `parse` function. It takes in a `LexerAnalyzer` and parses the token stream. If the parsing is successful, it returns a `DerivationTable` and a `SemanticStack`. Otherwise, it returns an error.

In the `ast.rs` file contains the code to handle semantic actions and the generation of an AST. A `Node` is composed of a optional `NodeVal` and a list of children `Nodes`. A `NodeVal` can either be a `Lead` that holds a `Token` or an `Internal` that holds one of many `InternalNodeTypes`. An `InternalNodeType` represents one of the many semantic concepts / node families that are represented by our grammar.

I also defined the `SemanticStack` struct that acts as a stack of `Nodes`. It defines different operations on the stack, which map 1-1 to the `SemanticAction` enum.

The `SemanticAction` enum defines the different semantic actions I have inserted into my grammar.

Due to Rust's borrowing and ownership model, I have decided to stray away from the proposed design in the lecture slides. I was still able to successfully model the attribute grammar in a way that is correct.

Utils

In the `utils.rs` file, I have written a function to write the `DerivationTable` into a Markdown file. I also have written a function to write the `SemanticStack` to a GraphViz file.

Use of Tools

- The Rust programming language: I chose to implement the compiler in Rust. Rust is a multi-paradigm programming language that guarantees memory safety, is relatively fast (similar to C/C++) and provides powerful tools to build a compiler (pattern matching, algebreic data types, etc...). I also had a bit of familiarity with the language and wanted to further my own proficiency in it.
- Crates (Rust equivalent of libraries):
  - Regex: This crate provides a library for parsing, compiling, and executing regular expressions, which is useful when testing a lexeme for a token.
  - Lazy Static: Since you cannot declare/initialize static file variables (like C++), this crate provides a macro that enables you to do so. I use this to mostly in conjuction with the `Regex` crate so that my regular expressions are precompiled and available to any file in my project.
  - StructOpt: This crate allows me to easily create a CLI for my compiler by simply defining a struct of possible command line arguments.
  - DotEnv: This crate allows the injection of environment variables by writing them in a .env file in the project root.
  - Env Logger: This crate allows for ergonomic logging with customization done through environment variables.
- Other tools:
  - CLion: My IDE of choice for writing Rust code.
  - Git/Github: VCS
  - VSCode: Mostly for text editing and visualizing non-text files.
  - Regex 101: To test my regexes.
  - GraphViz: For creating the Finite Automatons required for the assignment & generate a visual representation of ASTs.
  - UCalgary Grammar Tool: For help in transforming the grammar into LL(1).