CS 4386.001, Compiler Design, Fall 2022
Project Assignment #2

Phase 1 Due: 11:59pm, October 28
Phase 2 Due: 11:59pm, November 9
Submission: *.tar.xz (or *.zip) via elearning
Maximum points: 100

I Introduction

In Project 2, we will work on implementing our parser. We will scan a file of tokens, and translate it into an Abstract Syntax Tree. Project 2 is split into two phases. The first phase (due October 28) focuses on the grammar that includes statements within methods, and the second phase (due November 9) focuses on the rest of the language.

Project 2 is built on top of Project 1. Project 1 sample solution has been provided. You can use the sample solution as a reference to improve your Project 1, but you are expected to work on Project 2 from your own solution of Project 1.

II Example

The project 2 lab (Oct 10) will go through an example parser project. Several key parts of the example project have been placed into releases. These are as follows:

Unworking Expression Solution: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_Error1

Working Binary Expressions: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_BinaryExpr_Checkpoint

All Statements Working: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_AllStatements_Checkpoint

Shift/Reduce Conflict: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_ShiftReduce

Conflicting Productions Flattened: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_Flattened

Conflict Fully Solved: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_ConflictSolved

Full Grammar Implemented: https://github.com/pattersonzUTD/UTDLang--/releases/tag/Part2_Complete

III The Grammar to Implement (Phase 1)

Program	→	Stmts
Stmts	→	Stmt Stmts | λ
Stmt	→	if (Expr) { Stmts } IfEnd | while (Expr) { Stmts } | Name = Expr ;
	|	read (Readlist) ; | print (Printlist) ; | printline (Printlinelist) ;
	|	id ( ) ; | id (Args) ; | return ; | return Expr ; | Name ++ ; | Name -- ;
	|	{ Stmts } Optionalsemi
IfEnd	→	else { Stmts } | λ
Name	→	id | id [Expr]
Args	→	Expr , Args | Expr
Readlist	→	Name , Readlist | Name
Printlist	→	Expr , Printlist | Expr
Printlinelist	→	Printlist | λ
Expr	→	Name | id ( ) | id (Args) | intlit | charlit | strlit | floatlit | true | false
	|	(Expr) | ~ Expr | - Expr | + Expr | (Type) Expr |
	|	Expr Binaryop Expr | (Expr ? Expr : Expr)
Binaryop	→	* | / | + | - | < | > | <= | >= | == | <> | || | &&

Order of Operations:

(), []	Left to Right
(prefix)+, (prefix)-,~,++,--	Right to Left
(type cast)	Left to Right
*,/	Left to Right
+,-	Left to Right
<,>,<=,>=	Left to Right
<>,==	Left to Right
&&	Left to Right
||	Left to Right
?:	Right to Left

(Phase 2)

Program	→	class id { Memberdecls }
Memberdecls	→	Fielddecls Methoddecls
Fielddecls	→	Fielddecl Fielddecls | λ

Methoddecls	→	Methoddecl Methoddecls | λ
Fielddecl	→	Optionalfinal Type id Optionalexpr ;
	|	Type id [intlit] ;
Optionalfinal	→	final | λ
Optionalexpr	→	= Expr | λ
ddecl	→	Returntype id (Argdecls) { Fielddecls Stmts } Optionalsemi
Optionalsemi	→	; | λ
Returntype	→	Type | void
Type	→	int | char | bool | float
Argdecls	→	ArgdeclList | λ
ArgdeclList	→	Argdecl , ArgdeclList | Argdecl
Argdecl	→	Type id | Type id [ ]
Stmts	→	Stmt Stmts | λ
Stmt	→	if (Expr) { Fielddecls Stmts } IfEnd | while (Expr) { Fielddecls Stmts }
	|	Name = Expr; | read (Readlist) ; | print (Printlist) ; | printline (Printlinelist) ;
	|	id ( ) ; | id (Args) ; | return ; | return Expr ; | Name ++ ; | Name -- ;
	|	{ Fielddecls Stmts } Optionalsemi
IfEnd	→	else { Fielddecls Stmts } | λ
Name	→	id | id [Expr]
Args	→	Expr , Args | Expr
Readlist	→	Name , Readlist | Name
Printlist	→	Expr , Printlist | Expr
Printlinelist	→	Printlist | λ
Expr	→	Name | id ( ) | id (Args) | intlit | charlit | strlit | floatlit | true | false
	|	(Expr) | ~ Expr | - Expr | + Expr | (Type) Expr |
	|	Expr Binaryop Expr | (Expr ? Expr : Expr)
Binaryop	→	* | / | + | - | < | > | <= | >= | == | <> | || | &&

IV Goal and Submission

The goal is to be able to identify that a language is being parsed correctly and that the states are transitioning in an expected manner. We can do this by outputting a formatted version of the test files based on the abstract syntax tree that we created. If the abstract syntax tree is correct, then the output should be as expected. To show that the tree has the correct structure, make sure to have the formatted output use tabs to indent all new scopes (bodies of methods, while loops, and if statements), and wrap all "Expr" nodes with parenthesis. Example outputs have been provided for the example project. Several test files are provided for both phases. Run your implementation on these and upload your results along with a tar or zip of your project. In addition to the provided tests, more advanced tests will be run to grade Project 2 (in both phases).