

Compilers

Lab Session 1

Simple expression grammar

Expr.g4

```
grammar Expr;

s : e ;

e : e '*' e
  | e '+' e
  | INT
  ;

INT : [0-9]+ ;
WS  : [ \t\n]+ -> skip ;
```

main.cpp

```
// create a lexer that consumes the character stream
// and produces a token stream
ExprLexer lexer(&input);
antlr4::CommonTokenStream tokens(&lexer);

// create a parser that consumes the token stream,
// and parses it
ExprParser parser(&tokens);

// call the parser and get the parse tree
antlr4::tree::ParseTree *tree = parser.s();

// print the parse tree (for debugging purposes)
std::cout << tree->toStringTree(&parser) <<
std::endl;
```

Simple expression grammar

Expr.g4

```
grammar Expr;
```

```
s : e ;
```

```
e : e '*' e  
  | e '+' e  
  | INT  
  ;
```

```
INT : [0-9]+ ;
```

```
WS : [ \t\n]+ -> skip ;
```

main.cpp

```
// create a lexer that consumes the character stream  
// and produces a token stream  
ExprLexer lexer(&input);  
antlr4::CommonTokenStream tokens(&lexer);  
  
// create a parser that consumes the token stream,  
// and parses it  
ExprParser parser(&tokens);  
  
// call the parser and get the parse tree  
antlr4::tree::ParseTree *tree = parser.s();  
  
// print the parse tree (for debugging purposes)  
std::cout << tree->toStringTree(&parser) <<  
std::endl;
```

Simple expression grammar

Exercise 1

- Complete the expression grammar to handle other operators:
 - Division
 - Subtraction
 - Unary minus
 - Predefined functions (e.g. $\max(a,b)$, $\min(a,b)$, $\text{abs}(a)$, ...)
 - ...

Simple expression grammar (v2)

Expr.g4

```
grammar Expr;

s : e EOF ;

e : e MUL e // MUL is '*'
  | e ADD e // ADD is '+'
  | INT
  ;

MUL : '*' ;
ADD : '+' ;
INT : [0-9]+ ;
WS : [ \t\n]+ -> skip ;
```

main.cpp

```
// create a lexer that consumes the character stream
// and produces a token stream
ExprLexer lexer(&input);
antlr4::CommonTokenStream tokens(&lexer);

// create a parser that consumes the token stream,
// and parses it
ExprParser parser(&tokens);

// call the parser and get the parse tree
antlr4::tree::ParseTree *tree = parser.s();

// create a visitor that will evaluate the expression
Evaluator eval;

// traverse the tree using this Evaluator
int result = eval.visit(tree);
cout << result << endl;
```

Simple expression grammar (v2)

```
////////////////////////////////////  
// Sample "evaluator" using visitors  
class Evaluator : public ExprBaseVisitor {  
public:  
  
    // s : e EOF ;  
    antlrcpp::Any visitS(ExprParser::SContext *ctx) {  
        return visit(ctx->e()); // get value of child expression  
    }  
  
    // e : e MUL e | e ADD e | INT ;  
    antlrcpp::Any visitE(ExprParser::EContext *ctx) {  
        if (ctx->INT()) { // if this node has a child INT  
            return std::stoi(ctx->INT()->getText());  
        }  
        else {  
            int left = visit(ctx->e(0));  
            int right = visit(ctx->e(1));  
            if (ctx->MUL()) // if this node has a child MUL  
                return left*right;  
            else // must be ADD  
                return left+right;  
        }  
    }  
};
```

main.cpp

Simple expression grammar (v2)

Exercise 2

- Complete the grammar using what you did in exercise 1
- Extend the Evaluator visitor to handle the missing operators and compute the result in each case.

Summary

Key concepts learnt in this session

- How to write simple antlr4 grammars
- How to create a main program that calls a Lexer and a Parser to get a parse tree.
- How to traverse the parse tree using *Listeners*
- How to decorate the tree (store information associated to each node) using `ParseTreeProperty` maps
- How to traverse the parse tree using *Visitors* that return a result after each visit