Class 12: Grammars & Parsing

6.102 — Software Construction
Spring 2024

A grammar for arithmetic expressions

Exercise: yellkey.com/**TODO**

Nanoquiz: yellkey.com/**TODO**

Open `warmupTest.ts` and run it with `npm run warmupTest`

In the output, look for and compare:

- the parse tree
  - relate to the grammar at the top of `parser.ts`
- the abstract syntax tree (AST)
  - relate to the classes `Plus` and `Constant` in `IntegerExpression.ts`

Fill in the TODOs in `warmupTest` with input strings that produce different results:

- same AST but different parse tree
- same AST leaves (54, 2, 89 in that order) and expression value,
  but different parse tree and different AST
- same AST leaves and value, but parse tree with fewest possible `primary` nodes

Nanoquiz

- This quiz is just for you and your own brain:
  - closed-book, closed-notes
  - nothing else on your screen
- Lower your laptop screen when you're done

yellkey.com/**TODO**

Multiplication

yellkey.com/**TODO**

Today's starting code can handle addition of integers: `5+(2+3)`

We want to support multiplication too: `5*(2+3*4)`

In the grammar at the top of `parser.ts`:

- Create a `product` nonterminal
  - Don't forget to modify the `enum IntegerGrammar`
- `sum` should now be a sum of products
- `product` should be a product of primaries
- `npm run grammarTest`; does it display the right parse tree for `5*(2+3*4)`?

What does this grammar do with the input string `1+2*3`?

```
@skip whitespace {
    expr    ::= sum | product;
    sum     ::= primary ('+' primary)*;
    product ::= primary ('*' primary)*;
    primary ::= constant | '(' sum ')' | '(' product ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Pick one:

- good parse tree
- wrong parse tree (doesn't respect PEMDAS)
- parse error (grammar doesn't match entire string)

What does this grammar do with the input string `1+2*3`?

```
@skip whitespace {
    expr    ::= primary ([+*] primary)*;
    primary ::= constant | '(' expr ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Pick one:

- good parse tree
- wrong parse tree (doesn't respect PEMDAS)
- parse error (grammar doesn't match entire string)

What does this grammar do with the input string `1+2*3`?

```
@skip whitespace {
    expr    ::= sum;
    sum     ::= product ('+' product)*;
    product ::= primary ('*' primary)*;
    primary ::= constant | '(' sum ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Pick one:

- good parse tree
- wrong parse tree (doesn’t respect PEMDAS)
- parse error (grammar doesn’t match entire string)

Multiplication

yellkey.com/**TODO**

Today's starting code can handle addition of integers: `5+(2+3)`

We want to support multiplication too: `5*(2+3*4)`

In the grammar at the top of `parser.ts`:

- Create a `product` nonterminal
    - Don't forget to modify the `enum IntegerGrammar`
- `sum` should now be a sum of products
- `product` should be a product of primaries
- `npm run grammarTest`; does it display the right parse tree for `5*(2+3*4)`?

Now update `makeAbstractSyntaxTree` in `parser.ts`:

- the `if ... else if ...` needs a case for `Product`
- `npm run parserTest` to check the answer for `5*(2+3*4)`

```
@skip whitespace {
    expr ::= sum;
    sum ::= product ('+' product)*;
    product ::= primary ('*' primary)*;
    primary ::= constant | '(' sum ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Which of these would have to change (pick all that apply):

grammar   `makeAST()`   AST data type

to support this new feature:

variables

`5x + 3y`

```
@skip whitespace {
    expr ::= sum;
    sum ::= product ('+' product)*;
    product ::= primary ('*' primary)*;
    primary ::= constant | '(' sum ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Which of these would have to change (pick all that apply):

grammar   `makeAST()`   AST data type

to support this new feature:

curly braces (with same meaning as parentheses)

```
{5+3}*6
```

```
@skip whitespace {
    expr ::= sum;
    sum ::= product ('+' product)*;
    product ::= primary ('*' primary)*;
    primary ::= constant | '(' sum ')';
}
constant ::= [0-9]+;
whitespace ::= [ \t\r\n]+;
```

Which of these would have to change (pick all that apply):

grammar `makeAST()` AST data type

to support this new feature:

negative numbers (but not subtraction)

```
5 + -3
```