Checkin 5

Expr ⟶ *Expr* **plus** *Expr*
Expr ⟶ *Term*
Term ⟶ *Term* **times** *Term*
Term ⟶ *Factor*
Factor ⟶ **intlit**

Add Subtraction to this grammar. The new rule(s) should maintain arithmetic precedence and associativity. Also, make the grammar unambiguous.

Does this change whether the grammar is left-recursive, right-recursive, or recursive?

Administrivia

On Written Work…

Sorry for the confusion!

If you attend class when a written work is due, you DO NOT have to turn it in

Administrivia

Behold: The Project Oracle!

https://compilers.cool/oracles/o1

What's the format of output <x> ?

- Submit input to the Oracle

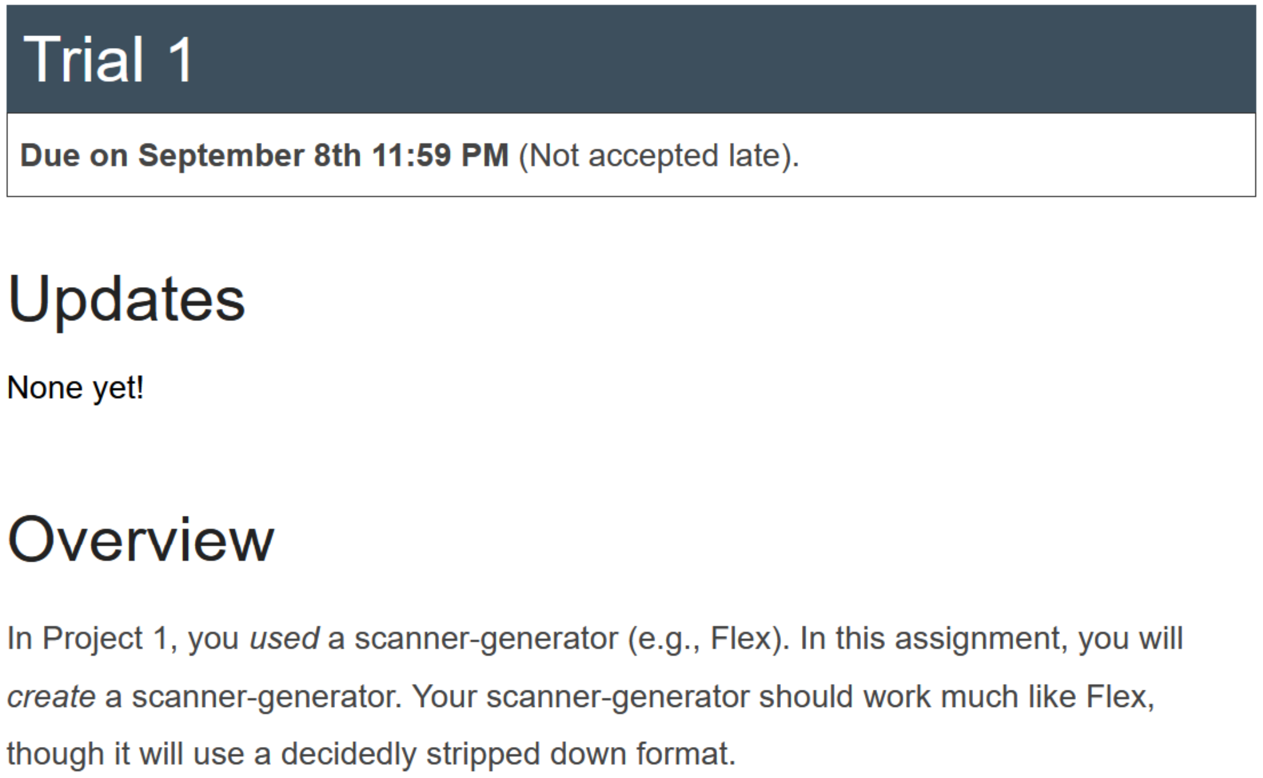
What's the token for character <y> ?

- Submit y to the Oracle

Administrivia

Behold: The Dragon Trials!

https://compilers.cool/trials/t1



Trial 1

Due on September 8th 11:59 PM (Not accepted late).

Updates

None yet!

Overview

In Project 1, you *used* a scanner-generator (e.g., Flex). In this assignment, you will *create* a scanner-generator. Your scanner-generator should work much like Flex, though it will use a decidedly stripped down format.

Flipped Wednesday

Written Work #1

Topics:

- Compiler Overview

Written Work #1: Question 1

What is the purpose of the lexer component of a compiler? Give an example of an input that GCC would flag for a lexical error.

Written Work #1: Question 2

What is the purpose of the syntactic analysis component of a compiler? Give an example of an input that GCC would flag for a syntactic error.

Written Work #1: Question 3

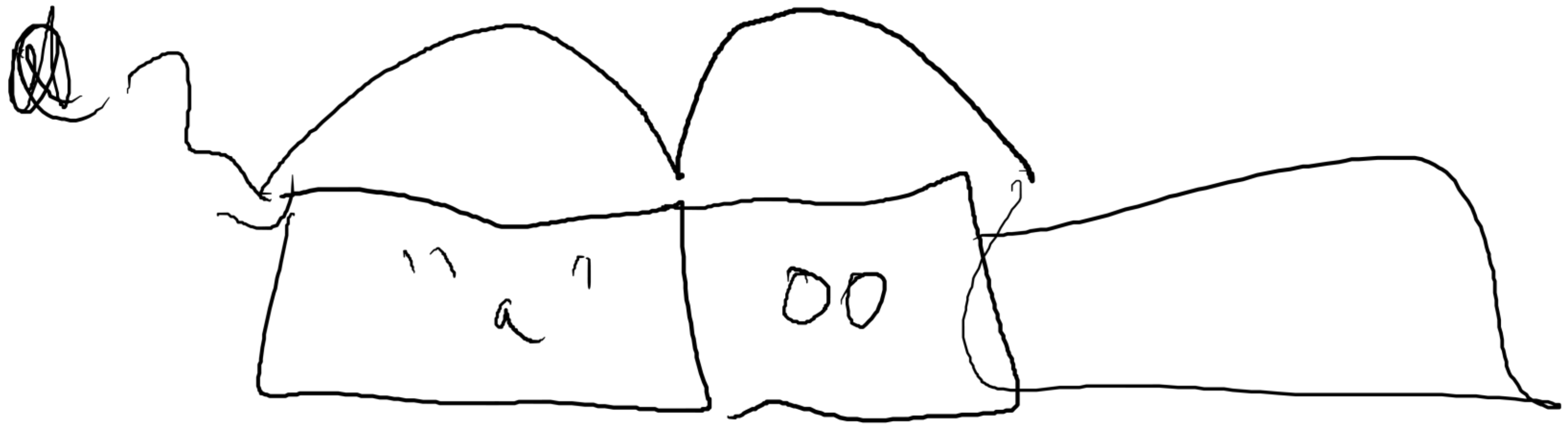
What is the purpose of name analysis in a compiler? Give an example of an input that GCC would flag for failing name analysis.

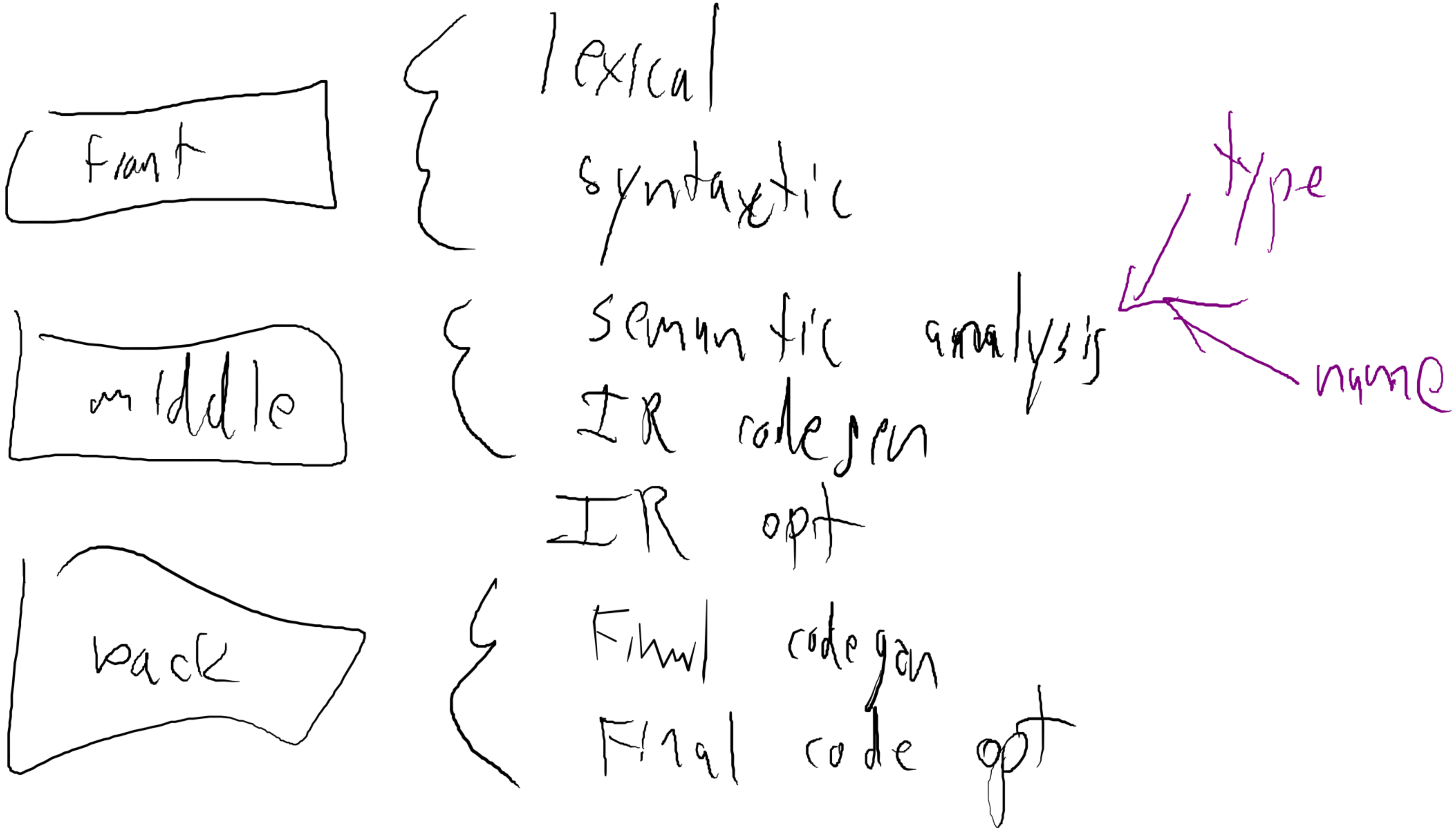
```
int main () {
    a = 4;
    int a;
}
```

```
int main () {
    if (true) {
        int a;
    }
    a = 2;
}
```

Written Work #1: Question 4

What is the purpose of type analysis in a compiler? Give an example of an input that GCC would flag for failing type analysis.




front
lexical
syntactic
middle
semantic analysis
type
name
IR codegen
IR opt
back
Final codegen
Final code opt

Create the full PEMDAS grammar

```
M → M * E
  | E
E → P ^ E
  |
P → ( S )
  | num
```

$3^{(5-6)}$

$1 \wedge (1+2)$