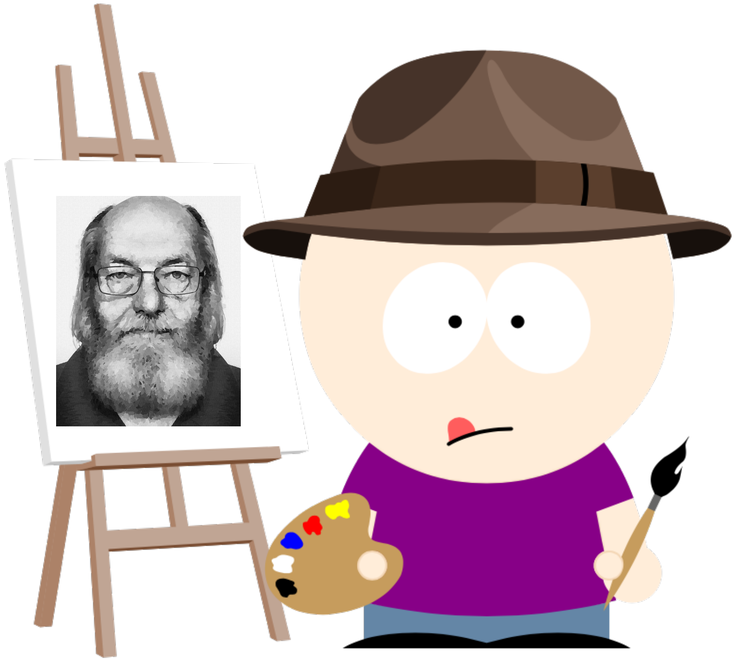
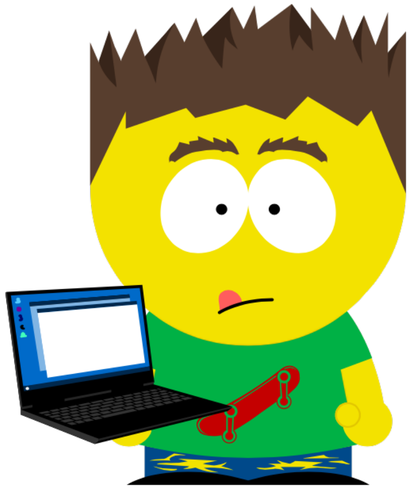




In-Class Exercises

Version: 2026-01-01

Parsing I - Grammars

Download exercises

1. Go to

   https://ligerlabs.org/compilers.html

2. Download the file

   `parse-1-exercises.zip`

3. Open up a terminal and Unzip the file

   ```
   > unzip parse-1-exercises.zip
   > cd parse-1-exercises
   > ls
   ```

Task 1

- What will the C program on the left, and the Python program on the right, print? Explain why!

```c
#include <stdio.h>
int main() {
    int a=1, b=2, c=3;
    if (c > b > a)
        puts("YES");
    else
        puts("NO");
    return 0;
}
```

```python
def main():
    a=1
    b=2
    c=3
    if c > b > a :
        print("YES")
    else:
        print("NO");

main()
```

Task 2

- What will this C program print? Explain why!

```
#include <stdio.h>
int main() {
    int a, x;
    x = 1 + (a = 10,20);
    printf("x=%i\n", x);
    return 0;
}
```

Task 3

- What will this C program print? Explain why!

```
#include <stdio.h>
int main() {
    int a=20, x=30;
    x -= a += 10;
    printf("x=%i\n", x);
    return 0;
}
```

Task 4

- What will this C program print? Explain why!

```
int x = 10;
int f1() {
   x++;
   return x;
}
int f2() {
   return x;
}
int main() {
   int a=f1() + f2();
   printf("a=%i\n",a);
   return 0;
}
```

- Yes, this is a trick question. If you don't know, Google "C order of expression evaluation" to figure out what's going on.

Task 5

- Starting with this ambiguous grammar:

```
E ::= E + E
      E * E
      E ^ E
      number
```

- Transform the grammar so that the following holds:
  1. "^", exponentiation, highest precedence, right associative
  2. "*", multiplication, medium precedence, left associative
  3. "+", addition, lowest precedence, left associative
- Examples:
  - a+b*c ⇒ a+(b*c)
  - a*b*c ⇒ (a*b)*c
  - a+b*c^d ⇒ a+(b*(c^d))
  - a^b^c ⇒ a^(b^c)

Task 6

- Transform the grammar from Task 5 so that all left recursion has been removed.

Task 7

- Consider this grammar:

$$
\begin{aligned}
S &\rightarrow S\ S\ \underline{+} \mid \\
&\quad\ S\ S\ \underline{*} \mid \\
&\quad\ \underline{a}
\end{aligned}
$$

1. Give a leftmost derivation and a parse tree for the string "a a + a *"
2. Give a rightmost derivation and a parse tree for the string "a a + a *"
3. Is the grammar ambiguous or unambiguous?

Dragonbook, Exercise 4.2.1

Task 8

- Consider this grammar for regular expressions:

Note: this is the vertical bar terminal symbol!

```
rexpr → rexpr | rterm | rterm
rterm → rterm rfactor | rfactor
rfactor → rfactor * | rprimary
rprimary → a | b
```

1. Left factor the original grammar.
2. Eliminare left recursion from the original grammar

Dragonbook, Exercise 4.3.1

Task 9

- Extend the Tiny parser so that it has 3 arithmetic operators:
  1. "^", exponentiation, highest precedence, right associative
  2. "*", multiplication, medium precedence, left associative
  3. "+", addition, lowest precedence, left associative
- You should make your changes in "task7-tiny/Parser.java"
- Base your implementation on the transformed grammar from Task 6.

Definitions and Algorithms

C Operator Precedence & Associativity

Operator	Kind	Prec	Assoc
a[k]	Primary	16	
f(· ··)	Primary	16	
.	Primary	16	
->	Primary	16	
a++, a--	Postfix	15	
++a, --a	Unary	14	
~	Unary	14	
!	Unary	14	
-	Unary	14	
&	Unary	14	
*	Unary	14	

Operator	Kind	Prec	Assoc
*, /, %	Binary	13	Left
+, -	Binary	12	Left
<<, >>	Binary	11	Left
<, >, <=, >=	Binary	10	Left
== !=	Binary	9	Left
&	Binary	8	Left
^	Binary	7	Left
|	Binary	6	Left
&&	Binary	5	Left
||	Binary	4	Left
? :	Ternary	3	Right
=, +=, -=, *=,	Binary	2	Right
/=, %=, <<=,			
>>=, &=, ^=, |=			
,	Binary	1	Left

Precedence & Associativity Transformation

1. Create non-terminals p_1, p_2, ..., p_n for each precedence level where p_n has the highest precedence level.

2. For operator op at precedence level i construct this production if the operator is left/right associative:

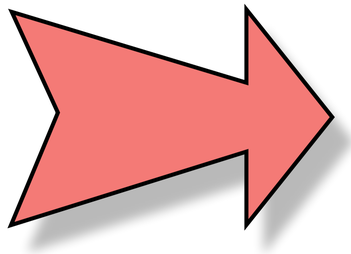
left associative	right assiociative
$p_i ::= p_i \text{ op } p_{i+1} \mid p_{i+1}$	$p_i ::= p_{i+1} \text{ op } p_i \mid p_{i+1}$

3. Construct a production for nonterminal p_{n+1} which represents primary expressions (identifiers, numbers, parenthesized expressions):

$$p_{n+1} ::= (p_1) \mid \text{num} \mid \text{id}$$

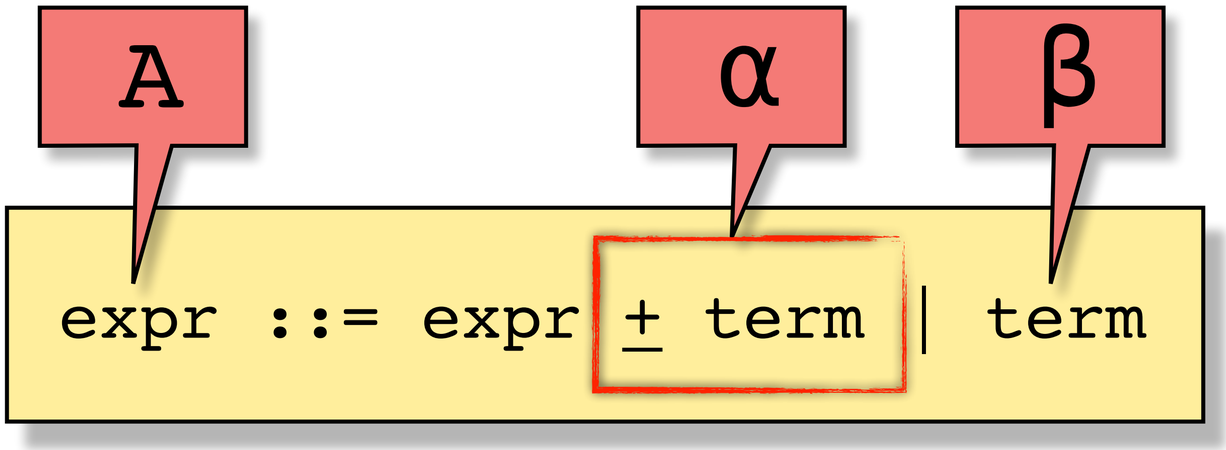
Left Recursion Removal

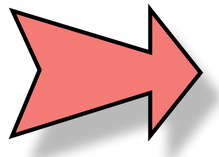
$$A \rightarrow A\ \alpha \mid \beta$$



$$A \rightarrow \beta\ R$$

$$R \rightarrow \alpha\ R \mid \epsilon$$





```
expr ::= term R
R    ::= ± term R | ϵ
```

collberg.cs.arizona.edu

ligerlabs.org

Supported by

NSF SATC/TTP-1525820

SATC/EDU-2029632

Christian Collberg

University of Arizona