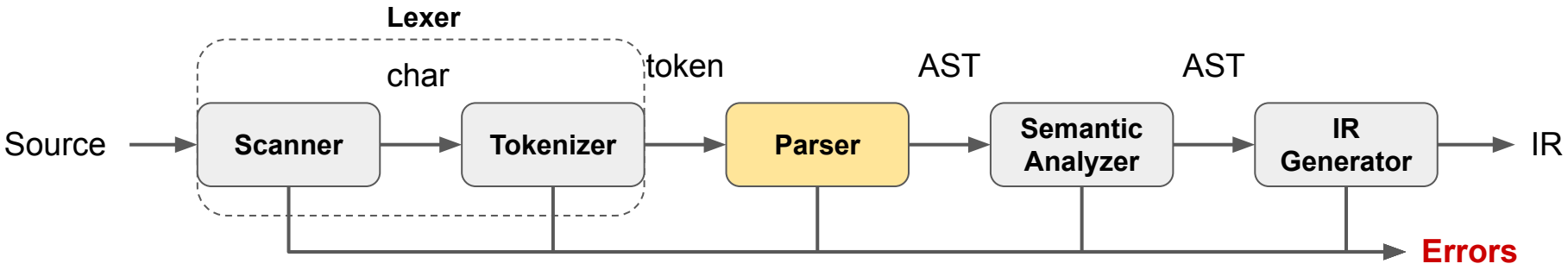
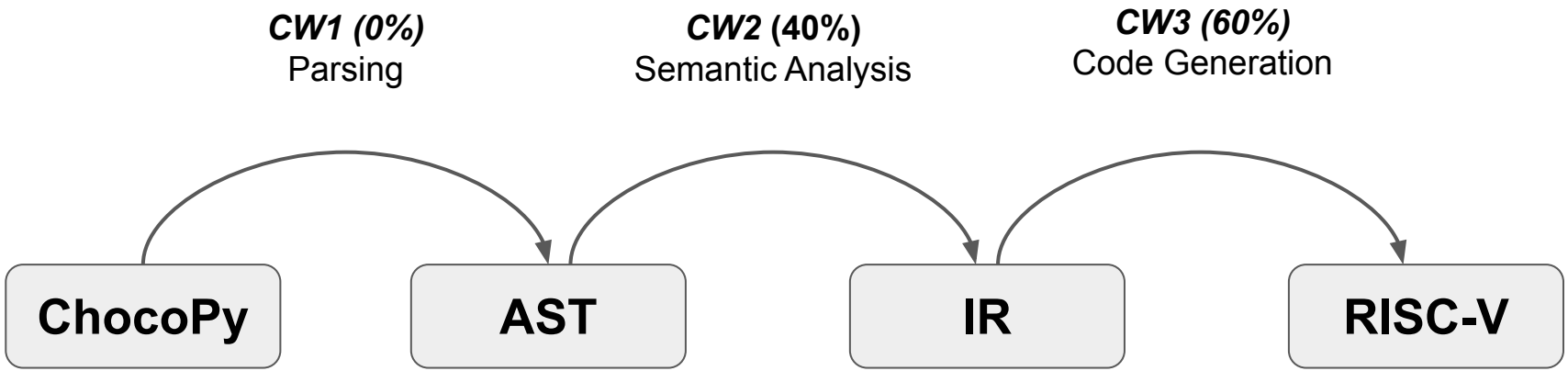
Compiling Techniques

Lecture 7: Coursework 1 - Intro

The Frontend



Coursework: A Python to RISC-V Compiler



Coursework Schedule

Week 1 (Jan 12)		Week 6 (Feb 23)	***CW2***
Week 2 (Jan 19)		Week 7 (Mar 2)	
Week 3 (Jan 26)	***CW1***	Week 8 (Mar 9)	
Week 4 (Feb 2)		Week 9 (Mar 16)	***CW3***
Week 5 (Feb 9)		Week 10 (Mar 23)	
Learning Week		Week 11 (Mar 30)	
		Week 11+1 (Apr 6)	

Deadlines: Friday noon

Coursework

“Check out Learn” → “Compiling Techniques” → “Assessment”

Provisional grades

Part 1: Parser

Test Case Statistics

	category	hidden	pass_public	fail_public	pass_hidden	fail_hidden
0	arithmetic-comparison-ops	15	15	0	15	0
1	combinations	13	0	0	11	2
2	complex-expressions	41	17	0	38	3
3	control-flow	6	3	0	4	2
4	function-prototypes	2	2	0	2	0
5	literals-identifiers	0	5	0	0	0
6	logical-conditional-ops	20	4	0	14	6
7	simple-statements	7	8	0	7	0
9	variable-defs-decls	10	7	0	10	0

Your result for part 1 is **64%** out of maximal 70%.

Part 2: Syntax Errors

Test Case Statistics

	category	hidden	pass_public	fail_public	pass_hidden	fail_hidden
8	syntax-errors	0	0	57	0	0

A recursive-descent parser

CFG for function call

```
funcall ::= ID ”(” arglist “)”
arglist ::= ID argrep | ε
argrep  ::= “,” ID argrep | ε
```

```
def parse_funcall():
    match(ID)
    match(LPAREN)
    parse_arglist()
    match(RPAREN)

def parse_arglist():
    if check(ID):
        match(ID)
        parse_argrep()

def parse_argrep():
    if check(COMMA):
        match(COMMA)
        match(ID)
        parse_argrep()
```

Parser Class

```
class Parser:

    def check(self, expected : TokenKind) -> bool:
        return self.lexer.peek().kind == expected

    def match(self, expected : TokenKind) -> Token:
        if self.check(expected):
            token = self.lexer.peek()
            self.lexer.consume()
            return token

       raise Exception(f“Error: token of kind ${expected) not found”)
```

What is a token?

A token consists of a token class and other additional information.

Example: some token classes

IDENTIFIER	→	foo , main , cnt , …
NUMBER	→	0 , −12, 1000 , …
STRING_LITERAL	→	”Hello world!”, ”a”, …
EQ	→	==
ASSIGN	→	=
PLUS	→	+
LPAR	→	(
…	→	…

```
class Token:
  Kind: TokenKind
  Value: Any = None
```