



UCF

College of Engineering
and Computer Science

UNIVERSITY OF CENTRAL FLORIDA

Generating Code for Arithmetic Expressions

COP-3402 Systems Software
Paul Gazzillo



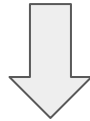
UCF

Expressions Use Intermediate Values

- Project 0: statements only have one operation
- Project 1: arbitrary expressions use many
- LLVM IR (and machine code): one operation at a time
- Compiler needs to emit and store each operation

Expressions Use Intermediate Values

```
print -5 + 2 * 3;
```



```
%t1 = mul nsw i32 2, 3  
%t2 = add nsw i32 -5, %t1  
call void @print_integer(i32 %t2)
```

Predictive Parsing Grammar

- Remove left recursion
- Parsing begins at starting symbol
- Parser choose a production at each step
- Parser uses a token lookahead to predict

```
expression  
  = term expression_prime
```

```
expression_prime  
  = PLUS term expression_prime  
  | epsilon
```

```
term  
  = factor term_prime
```

```
term_prime  
  = TIMES factor term_prime  
  | epsilon
```

```
factor  
  = LPAREN expression RPAREN  
  | NUMBER
```

Recursive Descent Parsing

- Each nonterminal is a function
- Each function body contains the productions
- Use lookahead to predict production
- Parse the production by either
 - consuming a token
 - calling the next nonterminal

```
expression():  
    term()  
    expression_prime()  
  
expression_prime():  
    if (next is PLUS):  
        consume PLUS  
        term()  
        expression_prime()  
    else:  
        // do nothing for epsilon  
  
factor():  
    if (next is LPAREN):  
        consume LPAREN  
        expression()  
        consume RPAREN  
    elif (next is NUMBER):  
        consume NUMBER  
    else: error()
```

Adding Code Generation

- Get parsing working first
- Add code that
 - Collects values
 - Generate temps
 - Emits code
- Create temps for nested expressions

```
expression():  
    left = term()  
    result = expression_prime(left)  
    return result
```

```
expression_prime(left):  
    if (next is PLUS):  
        consume PLUS  
        right = term()  
        result = newtemp()  
        emit result " = add " left ", " right  
        return expression_prime(result)  
    else:  
        // do nothing for epsilon  
        return left
```

Demo: Parsing and Code Generation

```
print -5;  
print -5 + 2;  
print -5 + 2 * 3;  
print (-5 + 2) * 3;
```