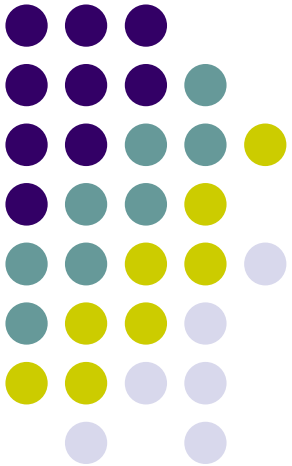
CS 460

Programming Languages

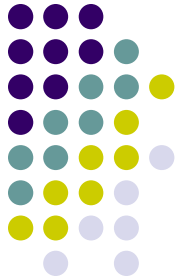
Fall 2023

Dr. Watts

(20 November 2023)

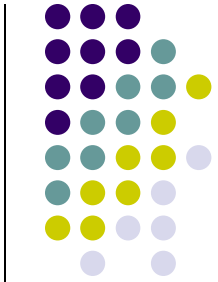


Assignments

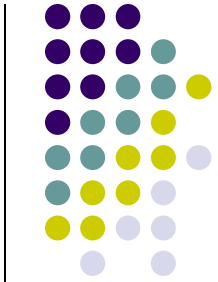


- Exercise 2
  - Script running so that your groups can improve their testing techniques
- Exercise 4
  - Spec now posted
- Project 3 and Exercise 5 will be posted this week.

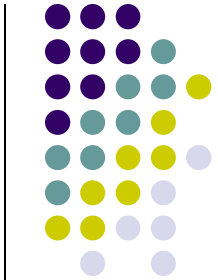
Project 2



- Spec and Framework posted
- Suggestions
  - Create First and Follow sets
  - Start with sets for the “Short Grammar”
  - Add in the remaining grammar rules
  - Testing
- Error Recovery
- Other Questions

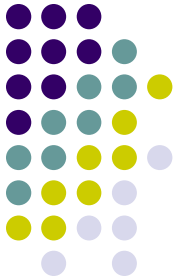


```
void SyntacticalAnalyzer::stmt ()
{
        if (token == NUMLIT_T || token == TRUE_T || token == FALSE_T
                                        || token == STRLIT_T)
        { // Rule 7  <stmt> -> <literal>
                lex->ruleFile << "Using Rule 7\n";
                literal ();
        }
        else if (token == IDENT_T)
        { // Rule 8  <stmt> -> IDENT_T
                lex->ruleFile << "Using Rule 8\n";
                token = lex->GetToken ();
        }
        else if (token == LPAREN_T)
        { // Rule 9  <stmt> -> LPAREN_T <action> RPAREN_T
                lex->ruleFile << "Using Rule 9\n";
                token = lex->GetToken ();
                action ();
                if (token == RPAREN_T)
                        token = lex->GetToken ();
                else
                        lex->ReportError ("( expected");
        }
        else
                lex->ReportError (lex->GetLexeme() + " unexpected");
        return;
}
```



```
void SyntacticalAnalyzer::stmt ()
{
        set <token_type> firsts = {IDENT_T, LPAREN_T, NUMLIT_T, STRLT_T,
                TRUE_T, FALSE_T,  SQUOTE_T, EOF_T, RPAREN_T};
        set <token_type> follows = {};
        while (firsts.find (token) == firsts.end())
        {
                lex->ReportError (“”);
                token = lex->GetToken()
        }
// The code from the previous slide
        while (follows.find (token) == follows.end())
        {
                lex->ReportError (“”);
                token = lex->GetToken()
        }
        return;
}
```

Project 3

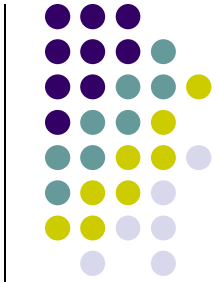


- PL460 to C++
- Code generation
- Start small
- P2-1.pl460

```
(define (main)
        (display "Hello World\n")
)
(main)
```

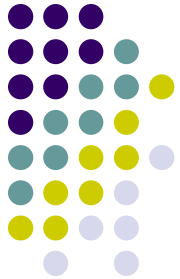
- What would this look like as a C++ program

Project 3



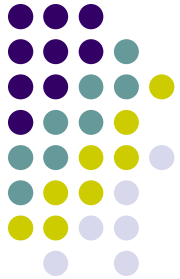
- Project 2 Syntactic Analyzer will make calls to Code Generator to write to .cpp file
- Look at the grammar
  - Insertion of calls to CodeGenerator
  - Where?
  - What strings should be written?
- <program> -> LPAREN_T <define> LPAREN_T <more_defines> EOF_T

Expressions and Assignment Statements (Chapter 7)



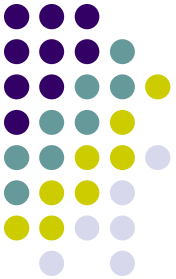
- Arithmetic Expressions
- Overloaded Operators
- Type Conversions
- Relational and Boolean Expressions
- Short-Circuit Evaluation
- Assignment Statements
- Mixed-Mode Assignment

Arithmetic Expressions



- Operators
- Operator Evaluation Order
  - Precedence
  - Commutativity
  - Associativity
  - Parenthesis
  - Conditional Expressions
  - Operand Evaluation Order
    - Side Effects

What is the output of this program?



```
#include <iostream>
using namespace std;

int main ()
{
        int a = 5, b = 7, c = 3;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "1. 5 + 7 * 3 - 3 * 5 % 7 --> ";
        cout << (5 + 7 * 3 - 3 * 5 % 7) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "2. 5 +- 7 * 3 - 3 % 5 *- 7 --> ";
        cout << (5 +- 7 * 3 - 3 % 5 *- 7) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "3. a + b * c - c * a % b --> ";
        cout << (a + b * c - c * a % b) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "4. a++ + b * c - c * a % ++b --> ";
        cout << (a++ + b * c - c * a % ++b) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "5. a += b * c - c * a % b --> ";
        cout << (a += b * c - c * a % b) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "6. a + (b * c) - c * (a % b) --> ";
        cout << (a + (b * c) - c * (a % b)) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        cout << "7. a + (b *= c) - c * (a %= b) --> ";
        cout << (a + (b *= c) - c * (a %= b)) << endl;
        cout << "a = " << a << "; b = " << b << "; c = " << c << endl;
        return 0;
}
```

What is the output of this program?

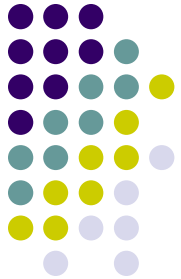
```
#include <iostream>
using namespace std;

int g = 10;
void reset (int & b)
{
        b = 7;
        g = 10;
}

int funky (int p, int & q)
{
        p = 2 * p;
        q = 1 + q;
        return (g = p + q);
}

int main ()
{
        int a = 5, b = 7;
        cout << "a = " << a << "; b = " << b << "; g = " << g << endl;
        cout << "1. funky (a, b) --> ";
        cout << (funky (a, b)) << endl;
        cout << "a = " << a << "; b = " << b << "; g = " << g << endl;
        reset (b);
        cout << "2. funky (a, b) + 2 * funky (a, b) --> ";
        cout << (funky (a, b) + 2 * funky (a, b)) << endl;
        cout << "a = " << a << "; b = " << b << "; g = " << g << endl;
        reset (b);
        cout << "3. 2 * funky (a, b) + funky (a, b) --> ";
        cout << (2 * funky (a, b) + funky (a, b)) << endl;
        cout << "a = " << a << "; b = " << b << "; g = " << g << endl;
        return 0;
}
```

Output . . . Why?



a = 5; b = 7; g = 10

1. funky (a, b) --> 18

a = 5; b = 8; g = 18

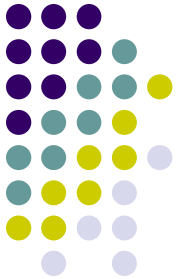
2. funky (a, b) + 2 * funky (a, b) --> 56

a = 5; b = 9; g = 19

3. 2 * funky (a, b) + funky (a, b) --> 55

a = 5; b = 9; g = 19

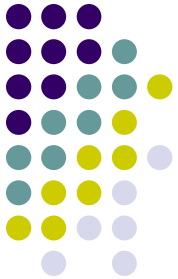
Overloaded Operators – Ex 5



- money operator + (const money & M) const;
- money operator += (const money & M);
- money operator - (const money & M) const;
- money operator -= (const money & M);
- money operator * (const double & F) const;
- **friend money operator * (const double & Factor, const money & M);**
- money operator *= (const double & Factor);
- money operator / (const double & Divisor) const;
- money operator /= (const double & Divisor);
- money operator % (const int & Divisor) const;
- money operator %= (const int & Divisor);
- money operator ++ (); // Pre increment
- money operator ++ (int); // Post increment
- money operator -- (); // Pre decrement
- money operator -- (int); // Post decrement

- bool operator == (const money & M) const;
- bool operator != (const money & M) const;
- bool operator < (const money & M) const;
- bool operator <= (const money & M) const;
- bool operator > (const money & M) const;
- bool operator >= (const money & M) const;

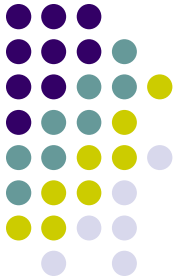
Overloaded Operators – Ex 5



- friend istream & >> (istream & ins, money & M);

- friend ostream & << (ostream & outs, const money & M);

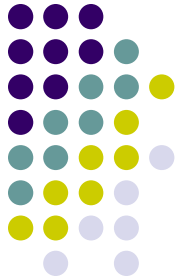
Overloaded Operators – Ex 5



- How do these differ?

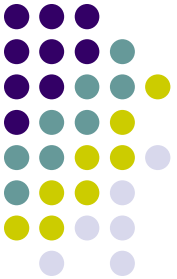
  - money operator * (const double & F) const;
  - **friend money operator * (const double & Factor, const money & M);**
  - money operator *= (const double & Factor);

Overloaded Operators – Ex 5



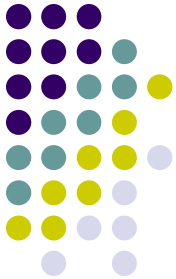
- How do these differ?
  - money operator ++ (); // Pre increment
  - money operator ++ (int); // Post increment
  - money operator -- (); // Pre decrement
  - money operator -- (int); // Post decrement

Relational and Boolean Expressions



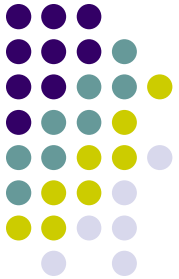
- if (a == b)
- cout << a == b << endl;
- Counting applications

Short-Circuit Evaluation



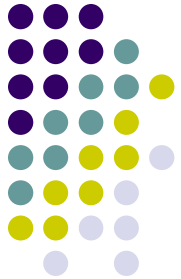
- if (a == b and c < d)
- if (a == b or c < d)
- if (function1 (a, b) and function2 (b, c))
- if (function1 (a, b) or function2 (b, c))
- Side effects
- if (letter == ‘a’ || ‘e’ || ‘i’ || ‘o’ || ‘u’)
- C++ vs Java

Assignment Statements



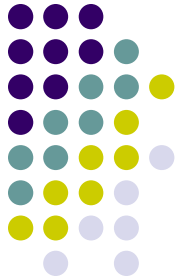
- As independent statements
- As part of an expression
- Return value

Type Conversions



- int a;
- float b;
- char c;
- Float (a);
- (unsigned short) c;

Mixed-Mode Assignment



- Coalescing / coercion
- In FORTRAN, C, and C++, any numeric value can be assigned to any numeric scalar variable; whatever conversion is necessary is done
- In Pascal, integers can be assigned to reals, but reals cannot be assigned to integers (the programmer must specify whether the conversion from real to integer is truncated or rounded)
- In Java, only widening assignment coercions are done
- In Ada, there is no assignment coercion