Recall

We use recursive descent parsing to go from program to AST (Abstract Syntax Tree).

Recall the Example Grammar

Recursion

The grammar drives the design of the parser

The AST drives the design of the code generator

Code for Expressions

Example Expression Code
Code For Assignment Statements

- Goal is to store the value of the expression into the variable (e.g., y)
  - We already have the code to place the expression’s value on top of the stack
  - Desired code:
    <code for expression>
    STOREOFF 1
- Example: y = x + 5;
  - PUSHOFF 0
  - PUSHIMM 5
  - ADD
  - STOREOFF 1

Code For Do Statements

- This is harder because we have to maintain a counter
- Goal is to
  - Code for expression
  - LOOP: DUP
  - NOT
  - JUMPC endloop
  - Code for statements:
    - PUSHOFF 0
    - PUSHIMM 1
    - SUB
    - JUMP loop
  - endloop: ADDSP -1

- Possible improvement: Code is wrong if <expression> is negative

Example Program and Resulting Code

```java
x = 1; y = 1;
do 5:
x = x * y;
y = y + 1;
end;
```

```
ADDSP 3
PUSHIMM 1
STOREOFF 0
PUSHOFF 0
PUSHIMM 1
TIMES
STOREOFF 0
PUSHOFF 1
PUSHIMM 1
ADD
STOREOFF 1
PUSHIMM 1
SUB
JUMP do0
end1: ADDSP –1
WRITE
WRITE
WRITE
STOP
```

EBNF

- BNF = Backus-Naur Form
  - A way of representing a grammar for a programming language
  - Originally Backus-Normal Form
    - Switched at suggestion of Knuth (partly because not really a normal form)
    - Naur was editor of Algol-60 document which used BNF
- EBNF = Extended BNF
  - Basically, BNF with some extra simplifying notation
  - There is an official standard, but common to modify it
- Typical constructs
  - Way to distinguish between terminals and nonterminals
  - * for repetition
  - [ ] for optional
  - { | } for choice

Example Grammar Notation: Java

```java
Statement:
Block
if ParExpression Statement [else Statement]
for ( ForInitOpt ; [Expression] ; ForUpdateOpt ) Statement
while ParExpression Statement
do Statement while ParExpression
try Block ( Catches | [Catches] finally Block )
synchronized ParExpression Block
return [Expression];
throw Expression;
break [Identifier]
continue [Identifier]
;
ExpressionStatement
Identifier : Statement
```
Example Grammar Notation: Python

```
if_stmt ::= "if" expression "then" suite
          ["elif" expression "then" suite]...
          ["else" "then" suite]
while_stmt ::= "while" expression "then" suite
             ["else" "then" suite]
for_stmt ::= "for" target_list "in" expression_list
           "then" suite
            ["else" "then" suite]
```

Grammar for Bali (Version for Part 2)

```
program -> int main( ) :
          [ declarations ] :
          statement* end
declarations -> type
             name
             ( type
              name )*
type -> ( int | boolean )
statement -> reference
           = expression
           ;
           statement
           |
           if
test
           then
           statement*
           ["else" statement*] endif
statement
           |
           loop
           statement*
           ( while | until ) test
           statement* endloop
statement
           |
           return expression
expression
           |
           print expression
           ( expression )*
reference
          -> name
expression
          -> [ + | - | not ] term
binaryOp
          -> arithmeticOp | comparisonOp | booleanOp
arithmeticOp
          -> + | - | * | / | %
comparisonOp
          -> < | <= | == | != | > | >=
booleanOp
          -> and | or
term
          -> literal | ( expression ) | inputValue | reference
literal
          -> integer | true | false
inputValue
          -> readInt
```

Rest of the Grammar for Bali (Part 2)

```
Coding Quality

- Pareto Principle
  - Named for Vilfredo Pareto, Italian economist, late 1800's
  - An 80/20 rule that shows up often
    - 80% of complaints are about 20% of the products
    - 80% of the decisions are completed during 20% of a meeting
  - Software version: 80% of software defects occur in just 20% of the modules
- NSA study (Drake, IEEE Computer, 1996) on 25 million lines of code
  - 70-80% of problems were due to 10-15% of modules
  - 95% of serious defects were from just 2.5% of the code
- Sturgeon's Revelation?
```

Unit vs. Integration Testing

- Unit testing
  - Testing of a single module
  - If a unit fails to match its specification then it is considered to be incorrect
- Integration testing
  - Testing of the entire program
  - Failure here may imply that the specifications are incorrect
  - Integration testing is usually harder than unit testing

Tools for Testing

- Goal: automate as much of the testing as we can
  - Some parts can't be automated
  - Process of developing test cases is difficult and usually cannot be fully automated
  - But we can automate the testing process itself
    - Both DrJava and Eclipse include facilities for using JUnit (http://www.junit.org)
  - Can make use of drivers and stubs
- A driver
  - Calls the unit being tested and keeps track of how it performs
- A stub
  - Simulates a program-part that is called by the unit being tested
- Both can interact with a file or with a person
  - Example: a driver can read calling parameters from a file and save test results to another file