Comp 401
Grammars, Recursive Descent

Instructor: Prasun Dewan
Prerequisites

- Inheritance Abstract Classes
# Predefined Course List

<table>
<thead>
<tr>
<th>Problems</th>
<th>Javadoc</th>
<th>Declaration</th>
<th>Console</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACourseDisplayer [Java Application] C:\Program Files\Java\jre1.5.0_04\bin\j</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please enter course title:
Intro. Prog.
TITLE NUMBER
Intro. Prog. COMP14

Please enter course title:
Comp. Animation
TITLE NUMBER
Comp. Animation COMP6

Please enter course title:
Lego Robots
TITLE NUMBER
Lego Robots COMP6

Please enter course title:
Meaning of Life

Sorry, this course is not offered.

Please enter course title:
Found. of Prog.
TITLE NUMBER
Found. of Prog. COMP114

Please enter course title:
static CourseList courses = new ACourseList();
static void fillCourses() {
    courses.addElement(new ARegularCourse("Intro. Prog.", "COMP", 14));
    courses.addElement(new ARegularCourse("Found. of Prog.", "COMP", 114));
    courses.addElement(new AFreshmanSeminar("Comp. Animation", "COMP"));
    courses.addElement(new AFreshmanSeminar("Lego Robots", "COMP"));
}
FILLING IT INTERACTIVELY?

Please enter course title:
Intro. Prog.
TITLE NUMBER
Intro. Prog. COMP14

Please enter course title:
Comp. Animation
TITLE NUMBER
Comp. Animation COMP6

Please enter course title:
Lego Robots
TITLE NUMBER
Lego Robots COMP6

Please enter course title:
Meaning of Life
Sorry, this course is not offered.

Please enter course title:
Found. of Prog.
TITLE NUMBER
Found. of Prog. COMP114

Please enter course title:
FILLING COURSE LIST INTERACTIVELY

Formal way of describing input syntax?
ANALYZING ENGLISH

<Sentence List> → <Sentence>*
<Sentence> → <Imperative Sentence> | <Interrogative Sentence> | ...
<Imperative Sentence> → <Transitive Verb> <Object>
<Transitive Verb> → study | play | ...
<Object> → grammars | tennis | ...

Correct Imperative Sentence?

study grammars play tennis play grammars study tennis tennis

Angle brackets?
TERMINALS VS. NON TERMINALS

Terminal is a token produced by the scanner (spell checker) that is not further expanded into other tokens – terminates

Set of all terminals is the vocabulary on which the grammar is based

Non terminal is a place holder for a legal sequence of tokens and is associated with rules describing how to derive these sequences
**Grammar: If-Else Statement**

```
if ( <Boolean Expression> )
    <Statement 1>;
else
    <Statement 2>;
```
Each program that processes input defines a grammar

- **String “RC”**
- **String “FS”**
- **Arbitrary int**
- **Arbitrary String**
- **Terminals (Tokens)?**
ANALYZING GRAMMARS

Production/Rule

LHS Root

Alternation

RHS

Repetition (0 or more)

<CourseList> → <Course> *.

<Course> → <RC> | <FS>

<RC> → RC <Title> <Dept> <Number>

<FS> → FS <Title> <Dept>

<Title> → String

<Dept> → String

<Number> → Integer

First

First will be used to decide on alternation
GRAMMAR-BASED CONCEPTS

Root: A distinguished non-terminal in the grammar that is either explicitly marked or does not appear on the RHS
- E.g. <Course List>, <Sentence>

Sentence: Any sequence of terminals
- E.g.: help offer, offer help, RC COMP 14 Intro. Prog. FC

Legal sentence: A sequence of terminals derived from the root using the grammar rules.
- E.g.: offer help, RC Intro. Prog. COMP 14 FS Random Thoughts

Legal phrase: A sequence of terminals derived from some non-terminal
- E.g. RC Intro. Prog. COMP 14

Language: The (possibly infinite) set of legal sentences
- E.g. {little boy, big boy, little girl, ...}

Parser: A program that recognizes a language (set of legal sentences), that is, can distinguish between a legal and illegal sentence and optionally produces a parse tree
Parser input and output

1. Scanner
2. Tokens
3. Parser
4. Parse Tree
5. Parse Tree Processor (Interpreter, Compiler, ...)

Concepts:
- Tokens
- Parse Tree
SCANNING VS. PARSING

Input Stream: Scanner Input

Token Stream: Scanner Input, Parser Input

Sentence list

Imperative Sentence

Transitive verb

Object

play
tennis

play
tennis

Consumed Input

Unconsumed Input

Consumed Tokens

Unconsumed Tokens

Parse Tree: parser output
PARSE TREE?

Course list

ARegularCourse

Title
Intro. Prog

Dept
COMP

Number
14

AFreshmanSeminar

Title
Random Thoughts

Dept
COMP
 COURSE PARSER NATURE

Scanner

Tokens

Parser

Parse Tree

Parse Tree Processor (Interpreter, Compiler, ...)

readString() and readInt() will be used to produce tokens

Will scan on demand rather than store in an array before parsing

Will learn general rules on how to write a parser
public class ACourseDisplayer {
    public static void main(String[] args) {
        fillCourses();
        ....
    }
}
**Main class: New Main method**

```java
public class ACourseDisplayer {
    public static void main(String[] args) {
        fillCoursesInteractively(); // parser method
        ...
    }
}
```

```
<CourseList> → <Course> *
<Course> → <RC> | <FS>
<RC> → RC <Title> <Dept> <Number>
<FS> → FS <Title> <Dept>
>Title → <String>
<Dept> → <String>
```
static void fillCoursesInteractively() {
    System.out.println("Please enter course info, terminating with a period:");
    courses = new ACourseParser().parseCourseList();
}

For each non terminal NT define a method parseNT() that returns a node in the parse tree

If NT¹ is on the LHS and NT² is on the RHS of a production then parseNT¹() calls parseNT²() to get a child node in is return value
package courses;
import collections.CourseList;
import collections.ACourseList;
public class ACourseParser implements CourseParser {
    //<CourseList> → <Course>*
    public CourseList parseCourseList() {
        CourseList courseList = new ACourseList();
        String nextToken;
        while (true) {
            nextToken = readString();
            if (nextToken.equals("."))
                break;
            else
                courseList.addElement(parseCourse(nextToken));
        }
        return courseList;
    }
}
// <Course> → <RC> | <FS>

Course parseCourse (String firstToken) {
    if (firstToken.toUpperCase().equals("FS"))
        return parseFreshmanSeminar();
    else if (firstToken.toUpperCase().equals("RC"))
        return parseRegularCourse();
    else {
        return null;
    }
}

Each method consumes and processes tokens in the token stream

First token consumed by the caller (parseCourseList()) and processed by this method (parseCourse())

Each method must know where in the token stream is the first unconsumed token

Instead of using Java Scanner to tokenize into strings and numbers, in general some custom scanner used to get tokens (terminals)
PROCESSING COURSE (Review)

```java
// <Course> → <RC> | <FS>
Course parseCourse (String firstToken) {
    if (firstToken.toUpperCase().equals("FS"))
        return parseFreshmanSeminar();
    else if (firstToken.toUpperCase().equals("RC"))
        return parseRegularCourse();
    else {
        return null;
    }
}
```

- Each method consumes and processes tokens in the token stream.
- First token consumed by the caller (parseCourseList()) and processed by this method (parseCourse()).
- Each method must know where in the token stream is the first unconsumed token.
- Instead of using Java Scanner to tokenize into strings and numbers, in general some custom scanner used to get tokens (terminals).
Course parseFreshmanSeminar() {
    String title = readString();
    String dept = readString();
    return new AFreshmanSeminar(title, dept);
}
PROCESSING REGULAR COURSE

//<RC> → RC <Title> <Dept> <Number>
Course parseRegularCourse() {
    String title = readString();
    String dept = readString();
    int number = readInt();
    return new ARegularCourse(title, dept, number);
}
**Recursive Descent Parsing**

For each non terminal, create a type, if needed, and a parse function that returns a value of that type, if a value is needed.

For each RHS non terminal or terminal, call the parse or scan function that returns corresponding value if needed.
Parser Structure

- Each production associated with a parser method.
- Parser method returns object associated with LHS of production if a return value expected.
- A parser method can recursively call parser methods to parse non-terminals in the associated production.
- A parser method for some production rule can call parser methods for non-terminals in the production, thereby “recursively descending” into lower levels of the tree deriving the input.
- The first token in the unparsed input is consumed and used to choose between alternatives.
- Such a parser called: recursive descent parser
- Illustrates top-down programming, next level details handled by called methods.
Why Grammars and Parsers

Precursor of some optional courses later

Understand the programming languages and program input

Recursive descent is important pattern for programming

Compilers, automata theory

Project

Recursion