CSE 131A
"Compiler Construction I"

Lecture 1

Scott B. Baden
Department of Computer Science & Engineering
University of California, San Diego
Winter 2007

Lecture 1

- General course information
- Course requirements and goals
- Course content overview
- Onyx and OXML
- Structure of modular translators
  - lexical analysis
  - syntactic analysis
  - semantic analysis and code generation

CSE 131A Staff

- Instructor: Scott B. Baden
- TA's: Brian Ryujin and Nakul Verma
- Tutors: Nes Fernando and Mooneer Salem
- All of us will see mail addressed here

cs131w@ieng6.ucsd.edu

General Course Information

- The course home page

  http://ieng6.ucsd.edu/~cs131w

- Bookmark it and read it!
  You're responsible for knowing the information there

Texts and readings

- Required text:

  Compilers: Principles, Techniques, and Tools, by A. V. Aho, R. Sethi, and J. D. Ullman. Addison Wesley (1988) **First edition**
- We won't be using the 2nd edition in this course
- Supplementary material will be available on line, linked to from the course web pages

Goals of the Course

- Learn general purpose techniques for understanding and building language translators together with theoretical underpinnings
- Apply the techniques to a real world problem: implementing a language for making database queries over collections of XML documents
- Software design
  - Automated software tools to enable us to focus on the intriguing parts of the problem
  - Managing a large software project
  - Language design and support issues

Prerequisite Background

- Basic automata and formal language theory
- Data structures and algorithms
- Programming
  Analyzing, Coding, Debugging, Testing

Course Requirements

- Four (4) programming assignments [60%]
  - May may be done in teams of 2
- Midterm exam [15%]
  - Thursday February 15, 2007
- Final exam [25%]
  - Comprehensive : all material presented in the course or appearing in your assignments
  - Thursday, March 20, 3:00 pm to 6:00 pm

Readings

- "What is XQuery?" (Bothner)
  http://www.gnu.org/software/qexo/XQuery-Intro.html
- Use and read about about RSS
  feed://www.nytimes.com/services/xml/rss/nyt/HomePage.xml

  http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html

Course Policies

- Academic Integrity
  - Cheating will not be tolerated
  - Familiarize yourself with the Academic Integrity Agreement http://www.cse.ucsd.edu/~baden/Integrity.html
- Late assignments will not be accepted

Computer labs

- The lab location will be posted later this week
- Home directory and web server
  ieng6.ucsd.edu
- Workstations available on campus
- If you have an OCE account, use it and run "prep cs131w" to set up your environment
- If you don't, request a "cs131w" account

About the discussion board

- Before posting a question, read the assignment description and associated documentation carefully
- If the question isn't answered there, **search** the webboard to see if your topic has been covered already
- When you post, write a descriptive subject line to help others search
- Include enough details in your message so we can determine what the problem is
- Post the minimal amount needed to illustrate the problem
- don't give away your own solution
- We may post email on the web board, with your name removed

Announcements

- Section
- Partners
- A0 (Onyx programming) due at
  9.00 PM on Wednesday 1/17

Next

- The Onyx programming language

Onyx

- Onyx is a language designed for convenience and power in manipulating collections of documents that conform to a subset of XML
- A subset of the emerging W3C standard for XQuery
- Onyx syntax permits simple expression of sophisticated queries and constructions of XML
- Though targeted to XML, Onyx is also a fully general programming language
- Particular features of Onyx make translation an interesting challenge

OXML

- Onyx manipulates structured databases represented in OXML
- Onyx eXtensible Markup Language for documents
- A data base is a *self-describing* set of data
- “Extensible” means there is neither a fixed document data structure nor typing
- The result is a flexible, powerful, standard for self-describing representation of structured data
- Many applications

  Consider RSS: “a format for syndicating news and the content of news-like sites”

Documentation

- Onyx Semantic Specification
  http://ieng6.ucsd.edu/~cs131w/Resources/Onyx_Spec/

- OXML and ODOM introductory document
  http://ieng6.ucsd.edu/~cs131w/Resources/XMLIntro.html

Onyx

- Onyx is an expression language
- An Onyx program returns a value:
  a sequence of expressions
- Every expression in turn has a value
- Onyx expressions have no side effects
- There is no updatable global state that persists across the execution of an Onyx program (though there are global definitions)

A first Onyx program

- There are no assignment statements in Onyx...
- ... instead, a value is bound to a variable with a **let** construct, which is like a “const” definition
- This Onyx expression has the value 56:

```
let $x := 5
   let $y := 6
      return 10*$x+$y
```

Conditional expressions

- Onyx has conditional ***expressions*** instead of conditional ***statements***
- Semantics similar to the ternary a?b:c operator in C/C++/Java
- Example

  ```
  if (3 < 4) then "yes!" else "no!"  ⇒ "yes!"
  ```

- What about

  ```
  if (3 > 4) then "yes!" else 0
  ```

Onyx data types

- Numeric types including Integer and Decimal
  3742   3741.33347   -6.6   -5
- Type Boolean represents logical values
  true and false
- Strings "Hello world!"
- Types for dealing with OXML documents

Sequences

- The comma operator is used to construct sequences

  3, 4, 2*2, "hello"

- Sequences cannot be nested
- It is legal to write sequences whose elements are sequences, but they are "flattened" when evaluated:

  ((1,2),3) = (1,2,3)

Example

- What does the following evaluate to?

```
let $a := (3,4)
let $b := ($a, $a)
let $c := “ninety”
return ( $a, $b, $c)
```

FLWR Expressions

- The fundamental control construct in Onyx
- FLWR is an acronym (pronounced *flower*) standing for the four possible Onyx subexpressions that it may contain:
  - **F**OR
  - **L**ET
  - **W**HERE
  - **R**ETURN
  - (We omit **O**RDER-BY)

The structure of a FLWR expression

- A FLWR expression consists of one or more **for** and/or **let** clauses …
  - followed by an optional **where** and
  - terminated by a mandatory **return**
- Here is a simple FLWR that has only a **for** and a **return:**

```
for $x in (1 to 3) return ($x,10+$x)
```

its value is

1 11 2 12 3 13

The structure of a FLWR expression

- Here is a simple FLWR that has only a **for** and a **return:**

```
for $x in (1 to 3)
    where ($x >2)
    return ($x,10+$x)
```

its value is

1 11 2 12 3 13

Core semantics of FLWR expression

- Where is redundant
- The following two code sequences are equivalent

```
for $x in (1 to 3)
    where ($x >2)
    return ($x,10+$x)
```

```
for $x in (1 to 3)
  return(
      if ($x >2)
        then ($x,10+$x)
      else ( )
  )
```

Creating OXML documents in Onyx

- The result of an Onyx program is an XML Document
- Onyx provides two kinds of operations to generate OXML
- Creation
  - Node constructors: **tnode()** and **enode()**
  - the builtin **document()** function
- Selection
  - Node deconstructors: **tagname()**
  - We will return to this subject later on

OXML Document Structure

- An OXML document is a tree structured collection
- A powerful way to structure data
  - Lists and tables are special cases of trees
  - We may also use a tree to represent graphs **http://www.w3.org/TandS/QL/QL98/pp/microsoft-serializing.html**
- The tree structure of an OXML document can be studied by looking at the grammar for OXML and at the Onyx Document Object Model (ODOM)
- Any document that can be derived from the OXML grammar is a *well-formed OXML document*

Syntax

- An OXML document consists of *elements* which can contain other elements, interspersed with character text data
- Text data is a string consisting of any characters **except** for < and &: **[^<&]***
- Elements are delimited by ***tags***
- Tags are in turn delimited by angle brackets ('<', ' >')

```
<head>
        <title>CSE 131A</title>
</head>
```

- The tags and text between <head> and </head> are the content of <head> element
- Tags must follow precise rules on structure and nesting

OXML Grammar

- The syntax of an OXML document can be (almost) defined using a BNF-like context-free grammar
- From the W3C XML 1.0 documentation, with some simplifications
- A document consists of a prolog followed by an element

  document ::= prolog element

- The element contains the document, the remaining parts contain additional information used to interpret the document

OXML Grammar: prolog

- The prolog must contain a declaration stating how the document is to be interpreted

- A standard declaration is:

```
<?xml version="1.0" encoding="utf-8" ?>
```

- This will be automatically taken care of by a library we'll provide

OXML Grammar: element

element ::= EmptyElemTag | STag content ETag

content ::= Text | Element*

Text ::= [^<&]*

- Exposes the recursive tree structure of an OXML document
- An element can contain other elements, interspersed with character text data

More of the grammar

- Content is **one occurrence** of **Text**

  abc xyz

- OR … **zero or more Elements**

  <a> <b1>abc</b1> <b1>xyz</b1>

- Mixed content with text and elements is *not* allowed

  **<a> <b1>NOT</b1> ALLOWED **

- Grammar

  content ::= Text | Element*

OXML Grammar: tags

EmptyElemTag ::= '<' Name (Attribute)* **'/>'**

STag ::= '<' Name (Attribute)* '>'

ETag ::= '</' Name '>'

- OXML tags delimit elements
- Since elements are nested, tags are nested
- The names in a STag and its corresponding ETag must match

  <review>Best ever.</review>

  <emptyTag/>
- A tag may contain zero or more **attributes**

Attributes

Attribute ::= Name '=' Quotes Attvalue Quotes

- Attributes are considered to represent *properties* of the elements in whose tags they appear; they are not part of the element's *content*
- No two occurrences of the same attribute name may appear in a given tag: an element's attributes form a set:

<element **symbol="Fe" num="56"**>Iron</element>

ODOM and the tree structure of OXML

- The tree structure of OXML is made explicit in the Object Document Object Model (ODOM)
- The ODOM is an abstract interface for manipulating XML documents
- We'll provide you with a library implementing the interface
- Objects are a subtype of Node
  - eNode or tNode
- For a given document, these Nodes exist in a tree structure that essentially follows the tree structure imposed on the document by the OXML grammar

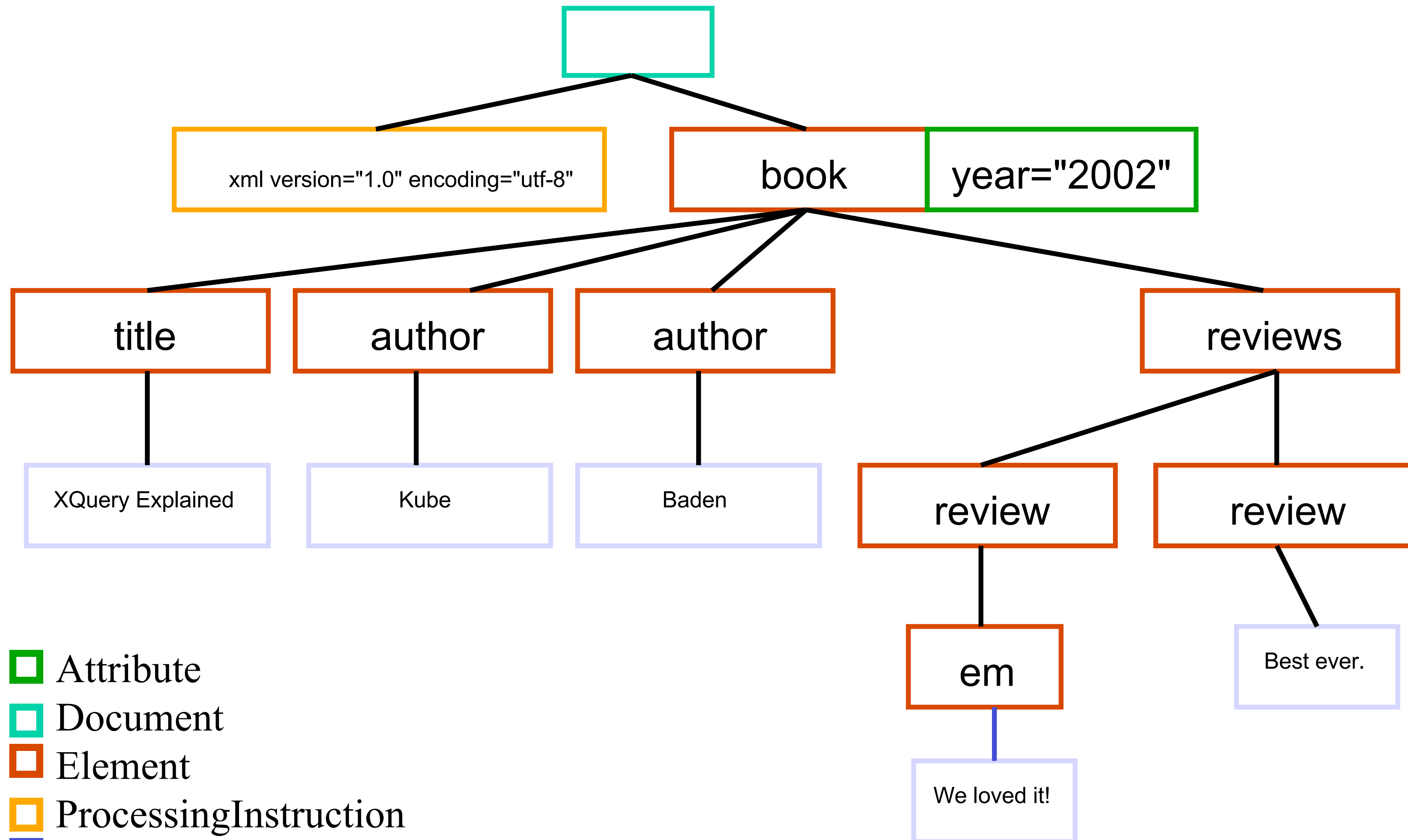
ODOM tree structure & document order

- The ODOM defines a tree structure for an OXML document
  - Attribute nodes "belong" to element nodes without being their children
- Customary tree relationships apply: root, parent, child, sibling, ancestor, descendant
- *Document order* for nodes in a ODOM tree corresponds to a preorder traversal of the tree
  - Attributes come after their element and before any children
- This is the same as the sequential order of the beginning of the corresponding items in the OXML document

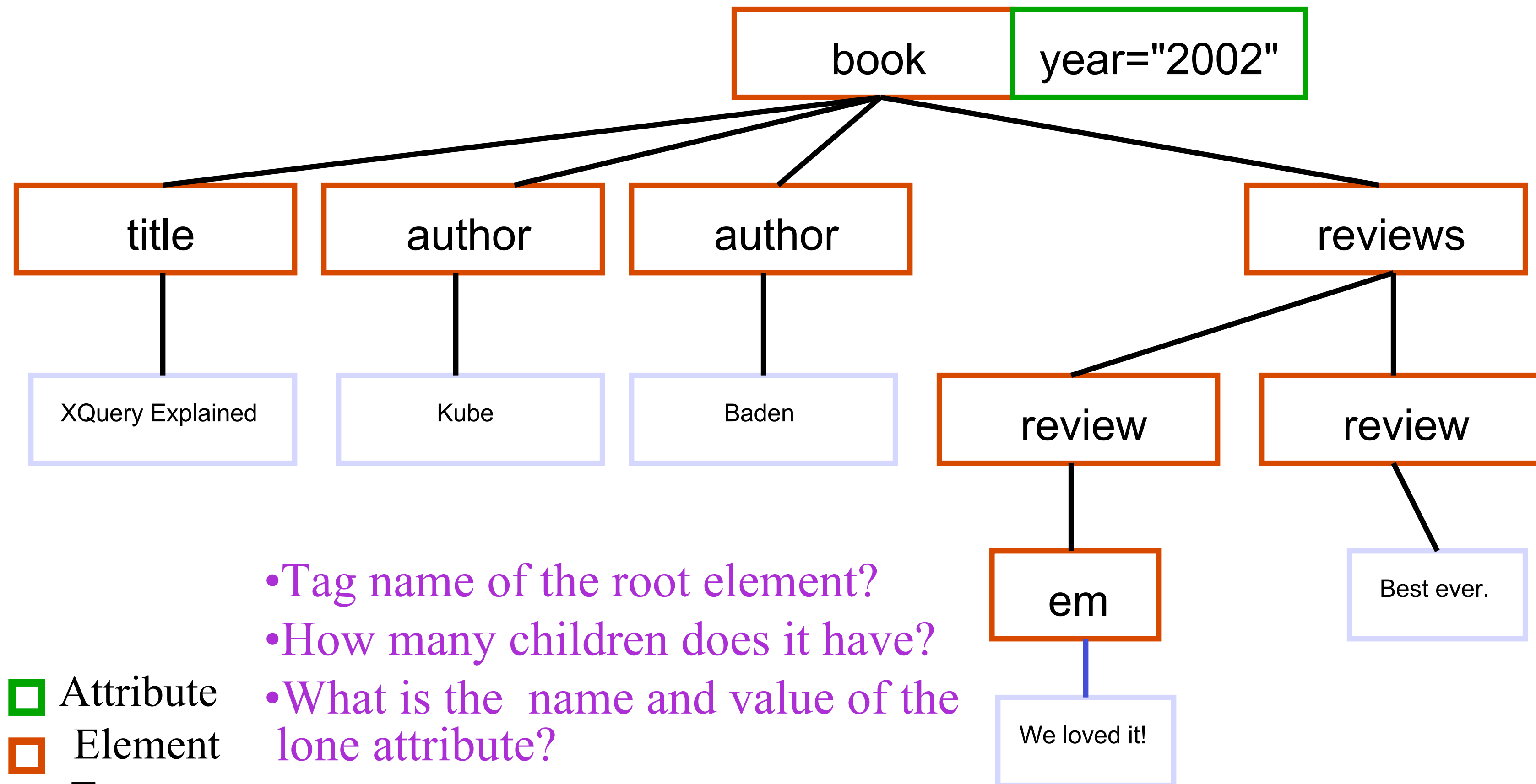
An OXML document

```
<?xml version="1.0" encoding="utf-8" ?>

 <book year="2002">

  <title>XQuery Explained</title>
  <author>Kube</author><author>Baden</author>

  <reviews>
      <review> <em>We loved it!</em></review>
      <review>Best ever.</review>
  </reviews>

 </book>
```

ODOM tree for the OXML document



ODOM tree for the OXML document



- Tag name of the root element?
- How many children does it have?
- What is the name and value of the lone attribute?

HTML and OXML

HTML

```
<HTML>
  <HEAD>
    <TITLE>CSE 131A</TITLE>
  </HEAD>
  <BODY>
    <H2>CSE 131A<BR>
     Fall 2002</H2>
    Welcome to CSE 131A!
  </BODY>
</HTML>
```

OXML

```
<?xml version="1.0"?>
<doc>
  <head>
    <title>CSE131A</title>
  </head>
  <body>
    <break/>
   <header>CSE 131A  Fall
                2002</header>
    <Greeting>
      Welcome to CSE 131A!
    </Greeting>
  </body>
</doc>
```

HTML and OXML

HTML

```
<HTML>
  <HEAD>
    <TITLE>CSE 131A</TITLE>
  </HEAD>
  <BODY>
    <H2>CSE 131A<BR>
     Fall 2002</H2>
    Welcome to CSE 131A!
  </BODY>
</HTML>
```

- html tags are standard
- Why isn’t this html document a well-formed XML document?

HTML and OXML

- Each well-formed XML file must begin with an **XML declaration**
- XML documents have no predefined tags
- The </break> tag has no embedded content
- Unlike HTML, XML embedded content is inferred from the syntax, not the element name

OXML

```
<?xml version="1.0"?>
<doc>
  <head>
    <title>CSE131A</title>
  </head>
  <body>
    <break/>
   <header>CSE 131A  Fall
               2002</header>
    <Greeting>
      Welcome to CSE 131A!
    </Greeting>
  </body>
</doc>
```

Formatting OXML

- White space is often needed to improve the readability of OXML
- We've provided xmlpp a pretty-printing filter and is available in the 131A environment (~/../public/Tools)
- Don't insert spaces, tabs, and other characters between element tags unless you intend to change the structure of the document
- Use diff to compare such output against a reference
- Use xmldiff to compare OXML documents