Code Translation
and
Introduction to Scheme

Judy Stafford
Comp 80 – Meeting 4
February 3, 2010

For today, lecture notes available at:
http://www.cs.tufts.edu/comp/80/lec

Agenda for the Day

- Administrative Stuff
  - Class list is okay
  - Book ready to float – anyone still need it?
  - Moodle…
- Three Approaches to Code Translation
- Introduction to Scheme
  - a “functional” language

2

Last time

- Syntax
  - **Problem**: how to precisely describe what a properly formed program looks like
  - Must cover all possible programs
- Solution: formal grammars
  - Inspired by work in linguistics
  - Notation: Backus-Naur Form (BNF)

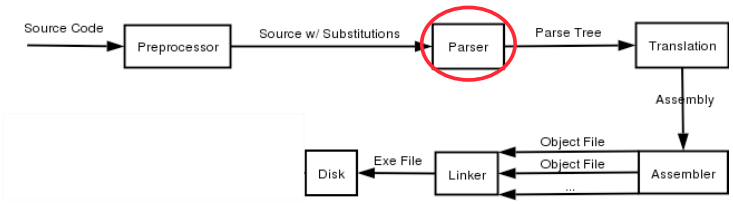
So, How is this used?

- Abstract syntax tree is the basis of "interpretation" and "compilation"
  - Interpreted languages
    - » Respond to you in real time
    - » You type in statement and get a result
      - or
    - » You feed it a list of statements in a file and get a result
  - Compiler
    - » You feed it a file with source code and it converts it to executable code that can be saved and run later

4

So, again, How is it used?

- Compilation (ex: C++) -- Five basic phases



- Interpretation (ex: Scheme) –
  - Each line is *parsed* into its smallest operations
  - Each operation is executed
  - Code remains as text – you just get the top line of the diagram above
    - » And, you get that every time you run the program!

5

How's Parsing done?

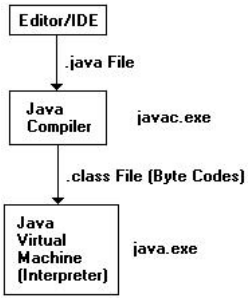
- Requires three phases
  - Lexer – read characters in and form tokens
    - » Requires a scanner and a tokenizer
    - » Produces a table of tokens
  - Syntactic analysis –
    - » Validates that tokens form a legal expression
  - Semantic analysis –
    - » Type checking
    - » Variable def/use relationships
    - » etc.

6

And then there is Java…

- Java code is “compiled” into byte code
- JVM (Java Virtual Machine) “interprets” the bytecode

“Write once, run anywhere”



7

Java Byte Code

- Example Java:

```
StringBuffer aStringBuffer = new StringBuffer("Hello");
aStringBuffer.append("World !");
```

- Example Byte Code:

```
0  new #8 <Class java.lang.StringBuffer>
3  dup
4  ldc #2 <String "Hello">
6  invokespecial #13 <Method java.lang.StringBuffer(java.lang.String)>
9  astore_110 aload_111 ldc #1 <String "World !">
13 invokevirtual #15 <Method java.lang.StringBuffer append(java.lang.String)>
16 pop
```

- The Byte Code is what is distributed and run
  - When it is run it is interpreted by the JVM for the target platform

8

Summarizing

- Grammars have been developed to define an unambiguous way for the machine to translate code
- Parse trees are used as an internal representation of the code
- Compilers and interpreters use parse trees to check the code and give feedback to programmer
- Finally the code is turned into executable in one of the three ways

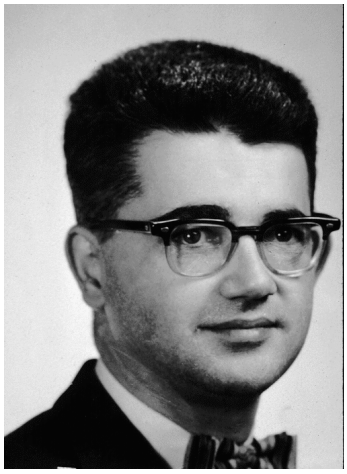
9

Intro to Scheme

- Scheme is a *functional* language
  - That doesn't mean it has more functions than C++ or Java
  - It means it is more like functions in mathematics

10

John McCarthy



- Pioneer in AI
  - Formalize common-sense reasoning
- Also
  - Proposed timesharing
  - Mathematical theory
- Lisp

  stems from interest in symbolic computation
  (math, logic)

Lisp, 1960

- Look at historical Lisp
  - Example of elegant, minimalist language
  - Not “C” (or Java): a chance to think differently
  - Many general themes in language design
  - Perspective
    - » Some old ideas seem new, but some are just old
- Many different dialects
  - Lisp 1.5, Maclisp, …, Scheme, ...
  - CommonLisp has many additional features
  - Use "Lisp" to mean family of languages
  - This course: we’ll use Scheme

Running programs

- A modern version of Lisp: **Scheme**
  - Interactive system: DrScheme
    Download from **www.drscheme.org**
  - It's available on linux machines
  - We'll do some Scheme programming
- Supports multiple dialects of Scheme
  - Click on "Language" tab then "Choose Language"
  - Choose "Advanced Student"
  - Use "Show Details" to set output style to "quasiquote"
- I'll demo as we go…

Languages you've seen

- Declaration **int x**
  - Introduces new identifier
  - May bind value to identifier, specify type, etc.
- Expression **(f(x) + 5) / 2**
  - Syntactic entity that is evaluated
  - Has a value, need not change accessible memory
    - » Change to memory is called a *side effect*
- Statement **p->next = q**
  - Imperative command
  - Alters the contents of previously-accessible memory

Languages you've seen

A program is…

- Sequence of statements
  - "Do this, then do this, then do this…."
  - Update the state of memory until we have the answer
  - Remember **x = x + 1** ?
- Possibly organized into procedures
  - For convenience
  - For reuse
  - Usually have "parameters" so that they can operate on different inputs

Scheme syntax: atoms

- Symbols

  foo bar course

  comp80 Computability

  1 2 3 7up
- Special symbols

  (For boolean truth values)

	TRUE	FALSE
Historical Lisp:	**T**	**nil**
Scheme:	**#t**	**#f**
DrScheme*:	**true**	**false**

*student language

Scheme syntax: lists

- Simple case: list of atoms

  (1 2 3 4)
  (apple orange banana)
  (2 apples 3 oranges no bananas)

  – What do you notice about the contents of lists?

  ➡ No distinction between kinds of atoms (no types)

- Lists can contain other lists

  ((2 apples) (3 oranges) (0 bananas))
  ((the (quick brown) fox) jumped over (the lazy dog))

- Empty list

  () ***or*** **nil** ***or*** **null**

Lists

- Very simple data structure
- BUT, you can build almost anything

- Examples:
  – A list (duh)
  – A record (or struct)
  – A list of records?
  – A binary tree?

Scheme programs

- Lists and atoms
  - Called "S-expressions"
  - S is for "symbolic"
  - That's it!
- OK, what's the trick?
  - Special way of interpreting lists as code – called *eval*
  - **Idea**:
    Given a list **(fn a1 a2 a3)**
    Treat **fn** as a function and **a1**, **a2**, and **a3** as arguments.
- **Notice**: lists used for both code and data

More details

- Function application

  (*fn* *arg*$_1$ ... *arg*$_n$)

  - First, evaluate each of the arguments
    (arguments might also be function applications)
  - Second, pass list of argument values to function
  - Notice: ***prefix*** notation – function name comes first
    (as opposed to ***infix*** notation: x + 3)
  - Many built-in functions
  - Later: make your own functions

Special forms: quote

- **Problem**:
  - What happens here: **(sum (1 2 3 4))**
  - What if I don't want to interpret the list as code?
- Quoting a value

  (quote *A*) → **A** *also written:* **'A**
  - Does not evaluate *A*
  - Just returns *A* as a value
- Examples:

  (quote 3) → **3**
  (quote (+ 3 2)) → **(+ 3 2)**
  '(+ 3 2) → **(+ 3 2)**
  '(To be or not to be) → **(To be or not to be)**

List manipulation

- Two main functions (built-in)
- **car** also called **first**
  - Returns the first element of the list
- **cdr** also called **rest**
  - Returns everything except the first element
  - Note: the result is always a list
- Use combinations to obtain parts

More lists

- How do I build a list?
- One option: use quote special form
  - What's the problem?
  - Can't compute the list parts:
    '(1 2 (first (3 4 5 6)) does not return **'(1 2 3)**
- **Solution**: **cons** function makes a new list
  - Takes an element and a list
  - Makes a new list with that element on the front
  - **(cons 1 '(2 3))** returns **(1 2 3)**
- Examples…

Other useful functions

- Tests (predicates)
  - **eq?, equal?** *Atom/List Equality*
    (eq? 3 4) → **false**
  - atom?, null? Is an atom or null?
    (atom? 'foo) → **true**
- Logical operators
  - and, or, not Logical operations
    (not true) → **false**
- How do we use these tests?

Interesting notes

- No assignment
  - Do we need it?
- Lists are never actually modified
  - We always get a new list
  - Sounds a little crazy, but it's very useful
- No loops
  - No for, while, goto, etc.
  - Problem for next time…
- Notice:
  - If we can build and manipulate lists
  - And lists are used to represent code…

Next time

- More on Scheme and intro to names and types
  - Read Chapters 4 and 5
- Short Scheme assignment due next Thursday
  - Will be posted on moodle asap
  - Will include instructions for downloading and running DrScheme