Discussion of Part 3 & Programming Languages

Lecture 7
CS 212 – Spring 2007

Announcements

Part 2 is due tomorrow (Thursday) evening
- We have an AMS.jar file (Assignment Management System) for submitting your assignment
- Instructions for its use will appear on the website later this evening

Do not alter the files we provide
- We run your code with the original versions of these files

Make use of Office Hours!
- If your Part 2 does not compile or if it fails many tests
  - The graders are not expected to determine the exact nature of any problems with your code
  - If there is some small error, you can request a regrade
    - Describe the problem
    - Describe the fix
    - Provide working code

Grammar for Bali (Part 3)

program → {declarations} | function*
function → type name ( {declarations} ) : statement* end
declarations → type name (, type name )*
type → int | boolean | void

- There must be a main function
- A function has a return-type and 0 or more parameters
- Valid types are int, boolean, and void
  - void can only be used as a function return-type

More Grammar for Bali (Part 3)

statement → reference = expression ;
statement → reference ;
statement → if expression then statement* [ else statement* ] endif
statement → loop statement* ( while | until ) expression ; statement* endloop
statement → print expression ( expression , expression )* ;
reference → name ( functionArgs )
functionArgs → ( ( expression , expression ) )

- The Part 3 sam-code for statements should be nearly the same as for Part 2
- The reference statement is executed for its side-effects (it might modify a global variable, for instance)
- To parse an assignment statement, pretend it's a reference statement until you reach the equal sign (=)

Rest of Grammar for Bali (Part 3)

expression → ( + | - | not | term ( binaryOp term )* )
binaryOp → arithmeticOp | comparisonOp | booleanOp
arithmeticOp → * | / | %
comparisonOp → < | <= | == | != | > | >=
booleanOp and | or
term → literal ( expression ) | inputValue | reference
literal → integer | true | false
inputValue → readInt

- The “hard stuff”
  - Implementing functions
  - Stack frames
  - Global variables
  - Use of multiple namespaces
  - Error handling
- Warning: finish this stuff before messing with the bonus work

The Major Tasks for Part 3

- Bonus work
  - Multiple error reporting
Global Variables & Namespaces

- At each point in the Bali-code, there are at most two active namespaces
  - A global namespace: Always exists, holds names for global variables and functions.
  - A local namespace: Exists only within a function, holds local variable names and parameter names.

- Each namespace corresponds in a natural way to a symbol table.
- To find a name your code should:
  1. First check the local symbol table
  2. Then, if not found, check the global symbol table.

- Once you have created the sam-code for a function, the function's symbol table can be discarded.

Recall: Stack Frames for Functions

- A new frame (on the stack) is created for each function call.
  - We use the FBR (Frame Base Register) to indicate the current frame.
  - The caller and the callee share responsibility for:
    1. Creating the stack frame
    2. Cleaning up the stack frame when the function is done.

- Once you have created the sam-code for a function, the function's symbol table can be discarded.

Signatures for Functions

- You need to check that each function argument is of the correct type.
  - To do this, you need to remember the function's signature.

  - A function signature includes:
    1. The function name
    2. The number and types of all parameters
    3. The return type of the function

  - The natural place to record this information is in the symbol table.
  - You can encode a function's signature in any way you want.

Bonus: Multiple Error Reporting

- Error Handling:
  - We will test your Part 3 compiler's response to errors in supplied Bali programs.
  - Two kinds of errors:
    1. Syntax errors: code that violates the rules of the Bali grammar.
    2. Semantic errors: code that violates the rules of Bali semantics.

- For bonus, use the MultipleBaliException class to accumulate and report multiple errors.

- Which kind of error (syntax error or semantic error) is easier to deal with if we're trying to accumulate all errors?

Reference Statement vs. Assignment Statement

- According to Bali’s grammar:
  - A reference statement and an assignment statement both start out looking like a reference.
  - No way to tell that you are parsing an assignment statement until you get to the equal sign (=).

- Suggestion:
  1. Start parsing as if you are parsing a reference.
  2. Once the reference is complete, you check for the equal sign (=) to see if within an assignment statement.
  3. If in an assignment statement:
     - You need to re-examine the AST you just built (for the reference) to see if it can be the target of an assignment statement.
     - Your compiler should throw a BaliSemanticException if the reference is inappropriate as a target.

So Many Languages

- Formula Translation (FORTRAN) in 1954 led to...
  - Over 2000 computer languages

- How many languages in use today?
  - Difficult to say
  - Legacy software (using outdated languages) is everywhere.

- Why can’t we just use one language?
Computer/Human Languages

- Computer/Human language matching game!

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Paragraphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens</td>
<td>Words</td>
</tr>
<tr>
<td>Token Separators</td>
<td>Chapters</td>
</tr>
<tr>
<td>Expressions</td>
<td>Book</td>
</tr>
<tr>
<td>Statements</td>
<td>Phrases</td>
</tr>
<tr>
<td>Functions</td>
<td>Alphabet</td>
</tr>
<tr>
<td>Classes</td>
<td>Sentences</td>
</tr>
<tr>
<td>Programs</td>
<td>Whitespace</td>
</tr>
</tbody>
</table>

Compiled vs. Interpreted

- Compiled
  - Parse code (typically create an abstract syntax tree)
  - Create assembly code for entire program
  - Run the assembly code

- Interpreted
  - Run each statement as the statement is parsed

Examples

- Compiled: Fortran, Java, C
- Interpreted: Matlab, Python, Logo, some versions of Basic

Advantages/Disadvantages?

Imperative vs. Declarative

- Imperative/Procedural
  - Statements tell the computer what to do
  - Think "commands" or "recipe"
  - Examples
    - Java, C, Fortran, Python

- Declarative
  - Describe what something "is like" (state what you know)
  - Examples
    - Logic programming (Prolog)
    - Constraint programming (later versions of Prolog)

Prolog Example

sendmore(Digits) :-
  Digits = [S,E,N,D,M,O,R,Y], % Create variables
  Digits :: [0..9], % Associate domains to variables
  S #\= 0, % Constraint: S must be different from 0
  M #\= 0,
  alldifferent(Digits), % All elements must take different values
  1000*S + 100*E + 10*N + D % Other problem constraints
  + 1000*M + 100*O + 10*R + E
  = 10000*M + 1000*O + 100*N + 10*E + Y,
  labeling(Digits). % Start the search

(from Wikipedia)

A List of Language Categories

- Procedural Language
- Imperative Language
- Declarative Language
- Applicative Language
- Functional Language
- Definitional Language
- Single Assignment Language
- Dataflow Language
- Logic Language
- Constraint Language
- Object-Oriented Language
- Concurrent Language
- Fourth Generation Language (4GL)
- Query Language
- Specification Language
- Assembly Language
- Intermediate Language
- Metalanguage

(from The Language List
(http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm))
Some Advice

• Use the language that best fits your task

• Think small!
  • Write little programs that test various concepts
  • Test them!
  • Comment them!
  • Collect these little programs together
  • Reuse your own code (templated!)