Week 9
Discussion of Assignment Part 3

Paul Chew
CS 212 – Spring 2004

Announcements

- No section tonight
  - Sections *will* be held next week (M & W)
- *Do not* alter the files we provide
  - Several groups altered the Part 2 files (e.g., some groups changed the packages)
- Make use of Office Hours!
- If your Part 2 did not compile or if it failed 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  ->  function*

function -> functionHeader functionBody

functionHeader ->
    ( type | void ) name ( [ parameters ] )

functionBody ->
    { variableDeclaration* } { statement* }

type -> ( int | boolean) [ [ ] ]

parameters -> type name (, type name )*

variableDeclaration -> type name ( , name )* ;
```

- There must be a *main function*
- Functions can be *overloaded*
- Arrays are one-dimensional
- Valid types are *int*, *boolean*, *int array*, or *boolean array*

More Grammar for Bali (Part 3)

```
statement ->  return [ expression ] ;
statement -> { statement* }
statement -> if expression then statement
    [ else statement ]
statement -> while expression do statement
statement -> do statement while expression ;
statement -> expression ;
statement -> print expression ;
statement -> ;
statement -> target = expression ;
target -> name [ subscript ]
```

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

Rest of Grammar for Bali (Part 3)

```
expression ->  expPart [ binaryOp expPart ]
expPart -> unaryOp expPart
expPart -> literal
expPart -> ( expression )
expPart -> name [ functionArgs | subscript ]
functionArgs -> ( [ expressionList ] )
expressionList -> expression ( , expression )*
subscript -> [ expression ]
literal -> integer | true | false | null
binaryOp -> arithmeticOp | comparisonOp | booleanOp
arithmeticOp -> + | - | * | / | %
comparisonOp -> < | > | <= | >= | == | !=
booleanOp -> && | || | ^
unaryOp -> - | !
```


The Major Tasks for Part 3

- The "hard stuff"
  - Implementing arrays
    - Use of the Heap
    - Use of **null**
  - Implementing functions
    - Stack frames
    - Overloading
  - Error handling
- **Warning**: finish this stuff before messing with any of the bonus work
- Bonus work
  - Multiple error reporting
  - Multidimensional arrays
    - Multiple subscripts
      - In declaration
      - In expression
      - In target for assignment statement
    - An additional kind of expression for array creation
  - Runtime error reporting

Code Patterns for Arrays

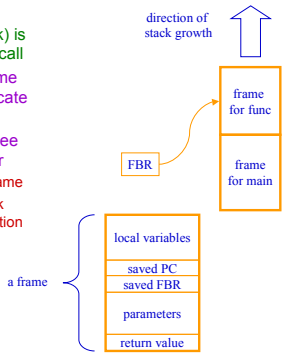
Bali Code	Sam Code	Comment
myIntegers = int[4];	PUSHIMM 4	*Push array's size onto Stack*
	MALLOC	*Create heap-block of size 5; push block's address onto Stack*
	PUSHIMM 1	
	ADD	*Address arithmetic*
	STOREOFF 13	*We arbitrarily assume myIntegers is at offset 13 from the FBR*
myIntegers[2] = 44;	PUSHOFF 13	*Push array's address onto Stack*
	PUSHIMM 2	*Subscript*
	ADD	*Address arithmetic*
	PUSHIMM 44	
	STOREIND	*Stores 44 into myIntegers[2]*
x = myIntegers[2];	PUSHOFF 13	*Push array's address onto Stack*
	PUSHIMM 2	*Subscript*
	ADD	*Address arithmetic*
	PUSHIND	*Stored value (44) placed on Stack*
	STOREOFF 9	*We arbitrarily assume x is at offset 9*

Use of null for Arrays

- Declaring an array
  int[ ] A;
- Constructing an array
  A = int[6];
- Initializing an array
  i = 0;
  while i < 6 do {
  A[i] = i;
  i = i + 1;
  }
- When an array is declared but not yet constructed, the array variable has value null
- In the sam-code, an array variable (e.g., A) holds the address of the array
  - After array construction, this is an address in Heap
  - Before array construction, this should be the address 0 (an address clearly not within Heap)
- In other words, null in Bali-code corresponds to 0 in sam-code

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
    - creating the stack frame
    - cleaning up the stack frame when the function is done



Recall: Signatures for Functions

- Functions in Bali can be *overloaded*
  - Functions can share same name as long as they differ in number or type of parameters
  - A function's *signature* determines which function to call
    - Signature encodes function's name as well as number and types of parameters
- Functions that share a name must *all* have same return type
- Bali does no automatic conversion of types
  - Thus function arguments and function parameters must match types exactly
- You can encode a function's signature in any way you want, but a Java String works fine

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
    - *Syntax errors*: code that violates the rules of the Bali grammar
    - *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?

Bonus: Multidimensional Arrays

- Multiple subscripts
- An additional kind of expression for array creation

Changes in grammar:

type -> (**int** | **boolean**) (**[ ]**)* *Multiple brackets now allowed*
target -> ***name*** [subscript*] *Multiple subscripts now allowed*
expression -> **[** [expressionList] **]** *A new kind of expression*
expPart -> ***name*** [functionArgs | subscript*] *Multiple subscripts now allowed*

Bonus: Runtime Error Reporting

- Examples
  - An attempt to divide by zero
  - An array subscript out of bounds
  - Using an array before its construction
- In general, there is no program that can reliably detect such errors at compile time (see CS 381/481: *undecidable problems*)
- These errors *can* be detected at runtime, but…
  - You have to check for them and generate error messages using sam-code

Recall: Expression Stmnt vs. Assignment Stmnt

- According to Bali's grammar
  - An expression statement and an assignment statement both start out looking like an expression
  - No way to tell that you are parsing an assignment statement until you get to the equal sign (=)
- Suggestion
  - Start parsing as if you are parsing an expression
  - Once the "expression" is complete, you check for the equal sign (=) to see if within an assignment statement
  - If in an assignment statement
    - You need to re-examine the AST you just built (for the expression) to see if it can be the target of an assignment statement
    - Your compiler should throw a BaliSyntaxException if the "expression" is inappropriate as a target