Week 8
Implementing Arrays

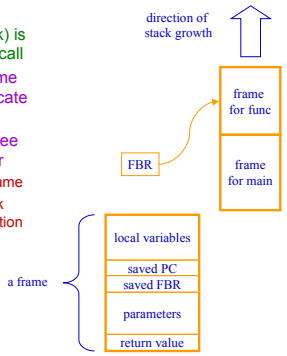
Paul Chew
CS 212 – Spring 2004

Announcements

- Late submission of Project Part 2
  - Allowed, but implies 20 point lateness penalty
  - Anyone who wishes may submit up until 11pm Thursday
  - Don't plan on late submission for remaining project parts
- Special section tonight
  - W evening, Mar 17
    - 7:30 to 8:20 PM
    - 205 Upson
    - David Levitan
  - Discuss Eclipse, an IDE for Java
  - Answer any remaining questions about Part 2

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



New in Part 3

- New data types *char* and *float*
  - Implementation is straightforward
  - Be sure to use the proper SaM instructions for floating point arithmetic
- Arrays
- Functions
  - Recursive (i.e., need stack frames)
  - Overloading
- Type casting
- Error handling

Arrays

- An array type is represented by a type followed by brackets
- Examples
  - int[ ] myIntegers;
  - char[ ] myCharacters;
- After these declarations, both myIntegers and myCharacters have the value *null*
- To initialize an array, assign an *array value*
- Array values
  - type[size]
    - Create array of given size
    - All elements have default value
  - [exp1, exp2, exp3]
    - Each expression
    - Creates an array of size = number of expressions
- Examples: both produce arrays of size 4 holding zeros
  - myIntegers = int[4];
  - myIntegers = [0, 0, 0, 0];

Multidimensional Arrays; Array Size

- Multidimensional arrays can be created by adding more brackets
- Example declaration
  - int [ ] [ ] values;
- Example initializations
  - values = [[1, 2, 3], [4, 5, 6]];
    - Produces 2-by-3 array of integers
  - values = [[1], [4, 5, 6]];
    - Produces Rows of varying length
- To determine size (number of elements) of an array
  - Each array has a size
  - Examples
    - myIntegers.size
      - Produces the value 4
    - values.size
      - Produces the value 2

Implementing Arrays

- Use the instruction MALLOC
  - Reserves space in the Heap
- Example sam-code

  PUSHIMM 4

  MALLOC
  - These instructions reserve a block of size 5 in SaM's Heap
    - 4 words for the array
    - 1 word to indicate the size of the block
      - In this case, the block size is 5
  - MALLOC leaves the block's *address* on top of the Stack
    - This is the address of the word that holds the block size
    - The array itself is located at *address*+1, *address*+2, *address*+3, and *address*+4
- The SaM Simulator does automatic garbage collection, so there is no need to free heap-space for arrays that are no longer in use

Overloading 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

Type Casting & Error Handling

- Type Casting
  - Bali allows some limited casting of types
  - An expression of the form *item.type* can be used to cast
    - float to int (decimal part is truncated)
    - int to float
    - char to int
    - int to char
  - There are sam-code instructions that do these conversions
- 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

No longer necessary

Expression Statement vs. Assignment Statement

- 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 can check for the equal sign (=)
  - 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