CS316 9-08-09

1. Brief Look At Java virtual Machine (JVM)
   (a) JVM Virtual Main Memory
      i. Function Code Area
         A. JVM Code for method functions
      ii. Runtime Stack – Control function calls
         A. At function call – a stack frame is pushed
         B. At function return – it's stack frame is popped
            - Stack Frames:
               - Stack Frame 1 - main
                  - var. area
                  - operand stack
               - SF 2 -
                  - var area
                  - operand stack
               - SF 3 -
                  - var area
                  - operand stack
               - SF 4 -
                  - var area
                  - operand stack
            - Var Area contains:
               - function parameters memory cells
               - Function local vars memory cells
            - Operation Stack (evaluation stack):
               - used to evaluate expressions
                  - example: x=E;
      iii. Heap Area
         A. Dynamically created arrays and class objects are collected here
         B. Garbage collector will collect unused array/class objects here
         C. Example Code:

```
void example()
{
        int i=0;
        while(i<100)
        {
                S;
                i++;
        }//while
}//example
```

            - The example function becomes (after Java compiler): {this sits in the Function Code Area}
              {left column is the virtual memory address}

```
0       iconst_0        //push constant 0 onto operand stack
1       istore_0        //pop top of stack and store into address 0 in var area
```

              {literally translates to “integer store into variable i”}

2 iload_0 //push value at address 0 onto stack
3 bipush 100 //push "byte integer" constant 0 onto stack {byte integer push}
5 if_icmpge 14 //pop top 2 values from stack; {"if (stack[top-1]>stack[top]) then goto 14"}
8+x iinc 0 1 //increment value at address 0 by 1
11+x goto 2 //
14+x return //

- Code for S goes between address 5 and 8+x
- x = the # of bytes taken by S

D. Every JVM instruction = 1 byte
E. An operand occupies 1 or 2 bytes depending on instructions
F. JVM is an example of stack-based VM. Uses operand stacks to evaluate expressions. No registers.
G. Another example: Common Intermediate Language, used for Microsoft's .NET framework

2. Compilation Stages:
   (a) High-level source program
   (b) Lexical analysis – can produce Lexical Errors (if illegal tokens are found)
      i. Extract tokens like identifiers, constant literals, delimiters/punctuation symbols, operators, keywords/reserved words
      ii. Tokens = lexical items, "lexemes"
      iii. void = keyword
      iv. example = identifier
      v. (= delimiter
      vi.) = delimiter
      vii. { = delimiter
      viii. int = keyword
      ix. i = identifier
      x. = = operator
      xi. 0 = constant literal
      xii. ++ = operator
      xiii. ; = delimiter
   (c) Stream of tokens
   (d) Syntactic analysis – can produce Syntax Errors (if grammatical errors are found) "Parsing"
      i. Analyzes grammatical structure and the identified structure is expressed in tree form
      ii. The precise description of grammar is given in context-free grammars
      iii. BNF (Backus-Naur Form) grammar
   (e) Parse tree
      i. Sample Tree:
      <function def>
      A. <return type>
         - void
      B. <function name>
         - example
      C. <parameter list>
         - empty

D. <body>
- <var declaration>
  - <var type>
    - int
  - <var name>
    - i
  - <assignment operator>
    - =
  - <expression>
    - 0
- <while loop>
  - ….....

ii. Type checking (for statically typed languages) – can produce type-checked parse tree
A. Type errors

(f) Semantic analysis – intermediate code generation
i. Semantic analysis checks non-grammatical properties

(g) Intermediate code