

# CS212

## SaM and Control Structures

1

## Generating Samcode

- Each node in AST usually represents some Samcode:
  - integers: **PUSHIMM *int***
  - variables: **PUSHOFF *address***
  - see templates document for suggestions/requirements
- Example:
 

```
(x + ( 2 + 3 ) )
PUSHOFF 291 // wherever x happens to be
PUSHIMM 2
PUSHIMM 3
ADD
ADD
```

2

## Assignment Statements

- Reminder from Chapter 1:
  - As you parse, you must keep track of variables
    - Each variable has space allocated on stack
    - Variable addresses must be remembered
  - Use **symbol table** (see **HashMap** in Java API)
- Pattern:
  - Grammar: **var = expr ;**
  - Samcode:
 

```
code for expr
STOREOFF varaddress
```
- Example: **x = ( 5 + y );** (assume x at address 1, y @ address 2)
 

```
PUSHIMM 5
PUSHOFF 2
ADD
STOREOFF 1
```

3

## Control Structures

- Samcode labels:
 

```
label:
instruction
or
label: instruction
```
- Provides mechanism for jumping to other instructions (how are labels represented internally?)
- To jump to another instruction:
 

```
JUMP label
```
- How to model selection and repetition?

4

## Selection Example: Bali--

Pattern for **if**-statement:

```
code for expr  
JUMPC label:  
code for when expr is false  
label:  
code for when expr is true
```

What does **JUMPC label** do?

sets **program counter** register to label  
so, SaM effectively jumps to the instruction with address label

```
main { { int rv ; int x ; int flag ; }  
  { x = 3 ;  
    flag = 0 ;  
    if ( ( 2 < x ) ) {  
      flag = 1 ;  
    }  
    rv = flag ; // return rv  
  }  
}
```

5

## Selection Example: Samcode

```
// Step 1: Start program and set variables  
ADDSP 3 // adjust SP to account for rv, x, and flag  
PUSHIMM 3 // push value of 3  
STOREOFF 1 // store the value 3 in address 1 for x  
PUSHIMM 0 // push the value of 0 (false)  
STOREOFF 2 // store the value 0 in address 2 for flag  
  
// Step 2: Check if 2 < x  
PUSHIMM 2 // push the value 2 to compare with x (Vbot)  
PUSHOFF 1 // push the value of x (Vtop)  
LESS // Push result of (Vbot < Vtop) to top of stack  
  
// Step 3: Process if statement  
JUMPC correct // check if result of GREATER is true (1) or false (0)  
// false:  
// if you had an else in Bali, you would handle it here  
JUMP continue // continue with remaining program  
correct: // true:  
PUSHIMM 1 // push the value 1 (true)  
STOREOFF 2 // store the value true for flag  
JUMP continue // continue with remaining program  
continue: // continue with program:  
PUSHOFF 2 // push the value of flag  
STOREOFF 0 // store the value of flag in rv  
ADDSP -2 // reset the SP  
STOP // done with the program
```

6

## Repetition

- Idea of **while ( expr ) statblock**:
  - if expression is true, do statement block
  - check expression again,
    - if true, repeat
    - otherwise, stop and resume rest of program
- Bali--** Example:

```
main {  
  { int x ; int rv ; }  
  {  
    x = 1 ;  
    while ( ( x < 5 ) ) {  
      x = ( x + 1 ) ;  
    }  
    rv = x ;  
  }  
}
```

7

## Repetition Samcode

```
ADDSP 2 // leave space for x and rv  
PUSHIMM 1 // push 1 on the stack  
STOREOFF 1 // store the value 1 for x  
looplabel: // label the loop starting at the condition  
PUSHOFF 1 // retrieve the value of x  
PUSHIMM 5 // push the value to compare x with  
LESS // is x < 5 ? push 1 if so; otherwise, 0  
JUMPC continue // if x < 5, do statements under continue  
done: // move to statement after the while-block  
PUSHOFF 1 // retrieve the value of x  
STOREOFF 0 // store the value of x as the rv  
ADDSP -1 // deallocate x  
STOP // stop processing and return the rv value  
continue: // the block of statements that follow the loop  
PUSHOFF 1 // retrieve the value of x  
PUSHIMM 1 // push 1 onto the stack  
ADD // add 1 to the current value of x  
STOREOFF 1 // store the new value of x  
JUMP looplabel // repeat the loop (goto loop condition)
```

8