Plan for Today

Finish accesses to member variables
- in general, how do we determine variable locations
- how will the LocalSTE for the implicit “this” parameter be created

General stack frame concept
- handling nested procedures
- questions answered by procedure call convention
- what about gcc for x86?

Determining locations for vars

Local vars
- maintain counter for method that is initialized to 0
- store counter in a temporary variable
- decrement current counter by size of the local variable
- return the value in the temporary variable

Class members
- maintain counter for method that is initialized to 0
- store counter in a temporary variable
- increment current counter by size of the local variable
- return the value in the temporary variable

Another example: where does each variable go?

class A {
    public static void main(String[] a) {
        System.out.println(42);
    }
}
class B {
    int[] x;
    boolean mBool;
    public int foo(boolean p1, int p2, B b, int[] y) {
        boolean v1; int i; int j; return 0;
    }
    public B bar() {
        B b;
        b = new B();
        return b;
    }
    public boolean baz() {
        B b; b = b.bar();
        return mBool;
    }
}

inAMethodDecl for BuildSymTable visitor

Steps needed in the inAMethodDecl
- does the method name conflict
- create a formal escape list
- add an entry into the formal escape list for the implicit “this” parameter
- create a list of types for explicit parameters
- for each explicit parameter add entry to formal escape list
- create Signature( return type, formal types list)
- create Frame with newFrame
- create MethodSTE and insert it into current ST
- push method scope
- create LocalSTE for implicit “this” and insert it into current ST
- create LocalSTE for each explicit formal and insert it into current ST
Nested Procedures Example

```c
float E(float x)
{
    float F(float y)
    {
        if (y<=0) {
            return 1;
        } else {
            return x + F(y-1);
        }
    }
    return F(2);
}
int main()
{
    printf(“%f
”, E(4));
}
```

Nested Procedures Suggested Exercise

```c
int foo(int x)
{
    int baz(int y)
    {
        return x*y;
    }
    int bar() {
        return baz(2);
    }
    int main() {
        printf(“%f
”, E(4));
    }
```

What is the output of the above program?
Draw the stack frame using static links.

Questions a calling convention must answer

Contract between caller and callee
- Where is the return value?
- Where is the stack pointer pointing upon entry to a function?
- Where are the parameters?
- Is the caller or callee responsible for popping the parameters?
- Does the stack pointer point at the top of the stack or the next empty slot?

Decisions needed for manipulating a frame/activation record
- Layout of callee-saved registers, caller-saved registers, locals, and temps
- Are parameters pushed by moving the stack pointer or is enough space set aside initially?

```c
int foo(int x, int y, int *z) {
    int a;
    a = x * y - *z;
    return a;
}
int main() {
    int x;
    x = 2;
    x = foo(4,5,&x);
    return x;
}
```
Suggested Exercise: funcCall1.c for x86

```c
#include <stdio.h>

int main(int argc, char **argv) {
    printf("Hello, World!\n");
    return 0;
}

void foo(int a) {
    printf("Hello, %d!\n", a);
}
```

What register holds the stack pointer? frame pointer? the return value? In instructions, where are source and dest? How is the local variable "a" accessed?