316 10-29-09

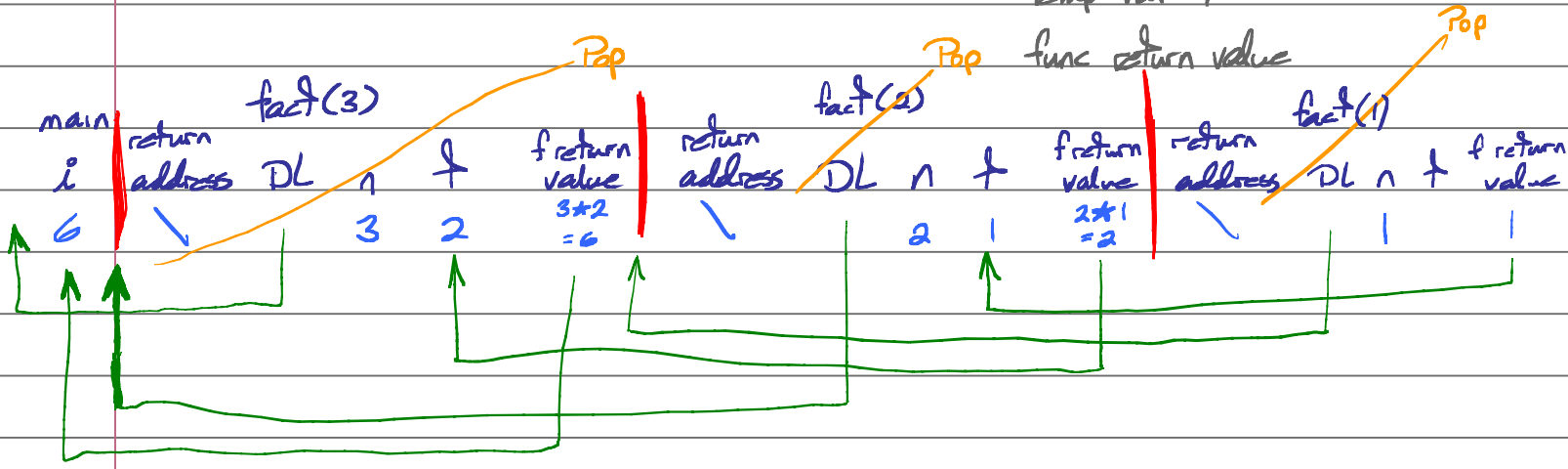
Note Title 10/29/2009

```
int fact(int n)
{ if(n<=1) return 1;
  else return n*fact(n-1);
}
void main()
{ int i = fact(3);
}
```

f = fact(n-1);
return n*f

AR for Main = var i

AR for Fact = return address
DL
parameter n
temp var f
func return value



Operational Semantics By intermediate Code

- Function Call - Assume call-by-value parameter passing
  - $a = f(E_1, \ldots, E_n)$

    code to evaluate E_1

    e_1 = result of above evaluation

    ⋮

    code to evaluate E_n

    e_n = result of above evaluation

    create new AR

    fill AR with necessary data

    AR.returnAdd = "returnAddress"

    AR.$x_1 = e_1$

    ⋮

    AR.$x_n = e_n$

    ⋮

    push AR onto runtime stack

    goto start_f

    returnAddress: a = AR.returnValue // not needed for T = void

    pop AR from runtime stack

Function Declaration

T f(x1, ..., xn) {... body ...}

start_f : code for body

Inside code for body

- Each return E; translates into:
  - code to evaluate E
  - AR.returnValue = results of above evaluation
  - goto AR.returnAdd; // goto the label contained in AR.returnAdd
- Each return; translates into:
  - goto AR.returnAdd
- IF there is no return statement, insert: goto AR.returnAdd at the end of body code.

Apply to fact function:

- combine operational semantics for if-else, function body, function call (inside)
- AR = the activation record for the function currently being executed
- AR' = the activation record for the called function to be pushed

```
start fact: code to evaluate "AR.n ≤ 1"
            b = result of above code
            if (b=0) goto Else;
            AR.returnValue = 1;
            goto AR.returnAdd;
            goto Out;
Else:       code to evaluate "AR.n-1";
            m = result of above evaluation;
            create new AR';
            fill AR' with necessary data
              AR'.n = m
              AR'.returnAdd = "return fact"
            push AR' onto runtime stack
            goto start fact
```

returnFact: AR.f = AR'.returnValue

pop AR' from runtime stack

code to evaluate AR.n * AR.f

AR.returnValue = result of above evaluation

goto AR.returnAdd

Out: