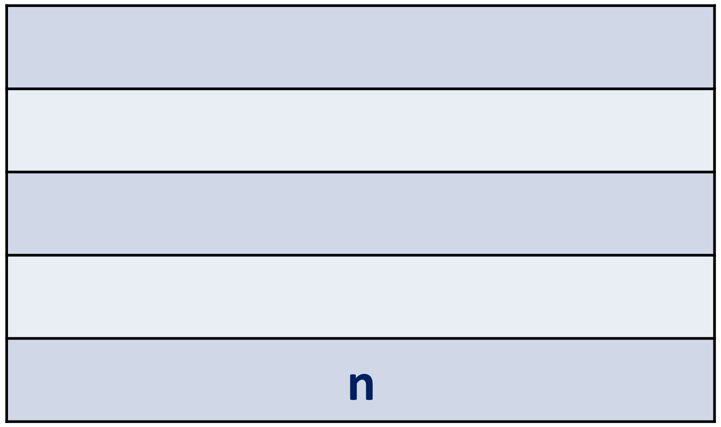
C to LC-3 Conversion – Recursive Running Sum

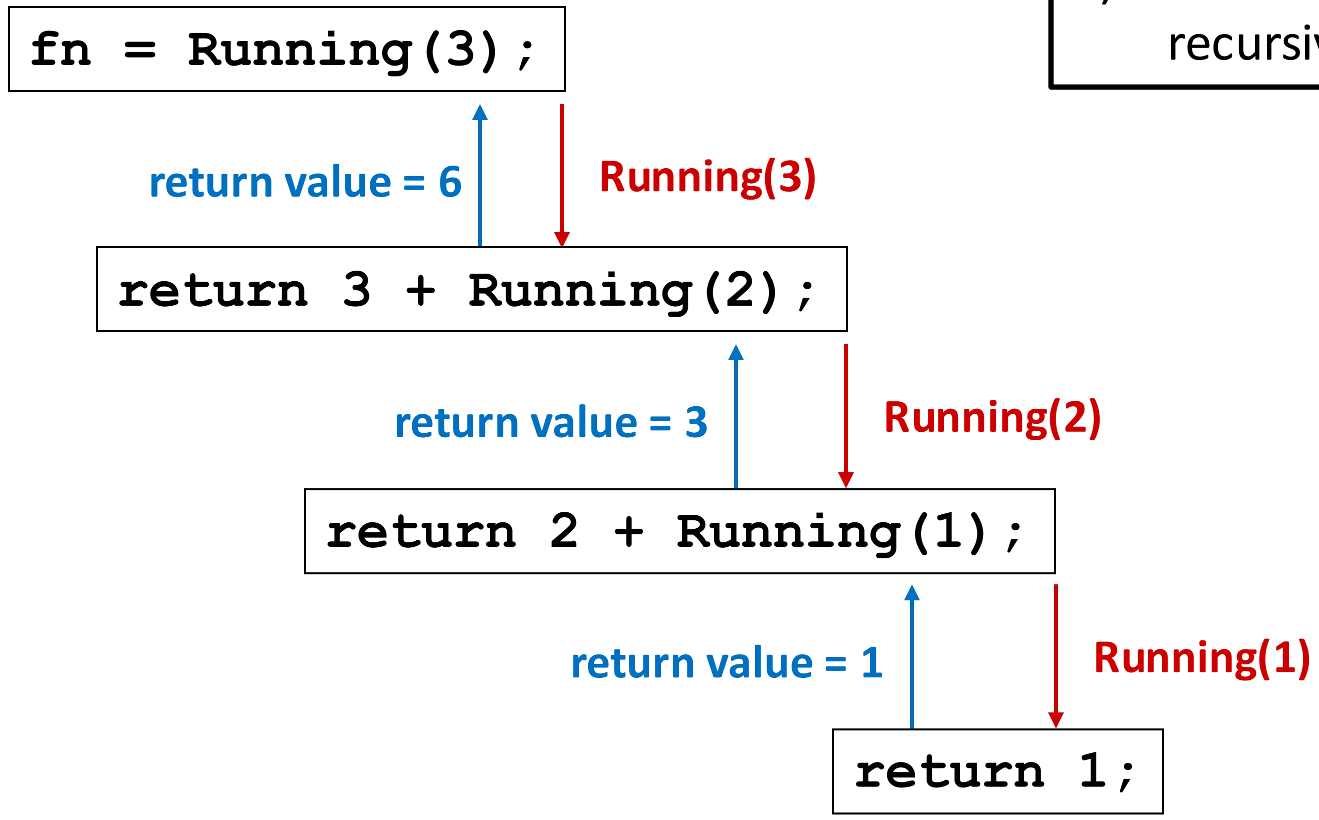
```
int Running(int n){
      int fn;
      if(n==1)
            fn = 1;
      else
            fn = n + Running(n-1);

      return fn;
}
```

Running Sum's Activation Record



ECE ILLINOIS

Executing Running Sum



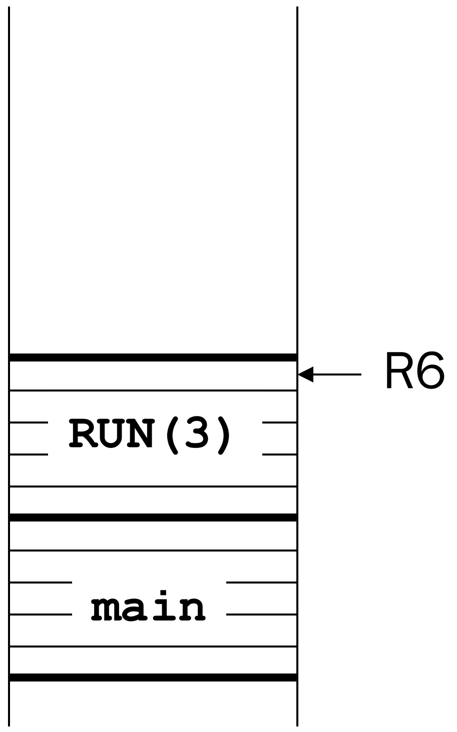
```
fn = Running(3);
return 3 + Running(2);
return 2 + Running(1);
return 1;
```

Observation:

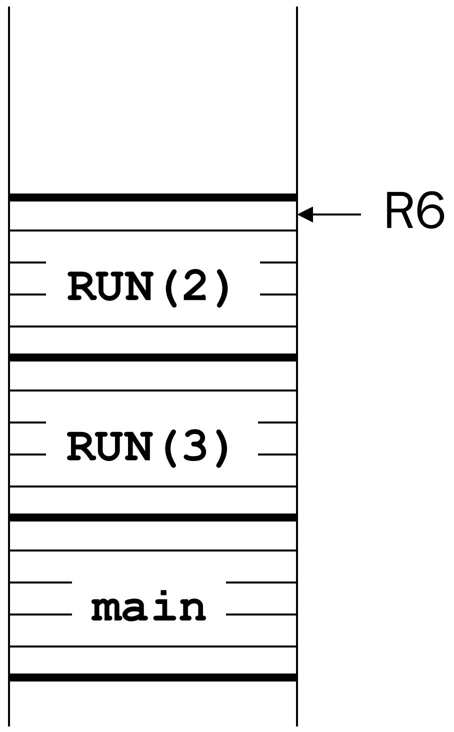
1) Each invocation solves a smaller version of the problem;
2) Once the base case is reached, recursive process stops.

Run-Time Stack During Execution of Running Sum

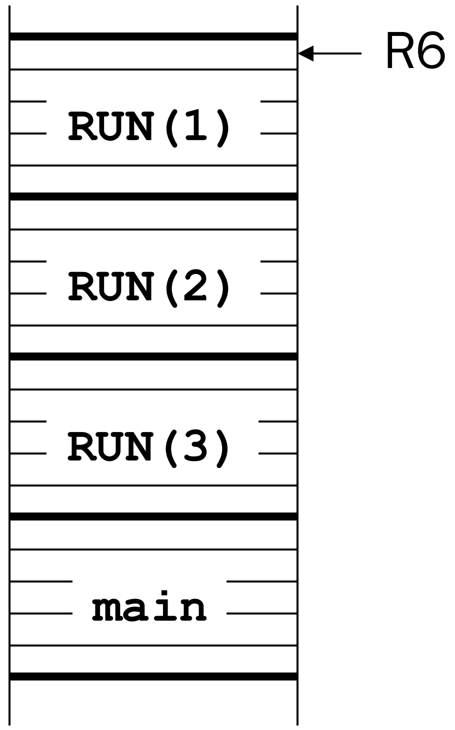
main → Running(3)



Running(3) → Running(2)



Running(2) → Running(1)



Stack Built-up and Tear-down

Caller function	1. **caller set-up** (push callee's arguments onto stack)
	2. **pass control to callee** (invoke function)
Callee function	3. **callee set-up** (push bookkeeping info and local variables onto stack)
	4. **execute function logic**
	5. **callee tear-down** (pop local variables, caller's frame pointer, and return address from stack)
	6. **return to caller**
Caller function	7. **caller tear-down** (pop callee's return value and arguments from stack)

```
;;convert Running Sum function to an LC-3 subroutine
RUNNING
;;callee set-up of Running(n)'s activation record
;push return value, return address & caller's frame pointer







;push local variables & update frame pointer




;;function logic
;base case (n==1)




BRz BASE_CASE
```

```
;;recursive case
;caller set-up for Running(n-1)
;push argument n-1 onto RTS



;call Running(n-1)



;caller tear-down for Running(n-1)
;pop Running(n-1)'s return value to R2




;pop Running(n-1)'s argument



;calculate n + Running(n-1)
```

```
;store result in fn


;ready to return


BASE_CASE
;set fn = 1





RETURN
;set return value
```

```
;;callee tear-down of Running(n)'s activation record
;pop local variables


;pop caller's frame pointer and return address




;;return to caller
```