Mc Graw Hill Education

Because learning changes everything.®

From Bits and Gates to C and Beyond

Recursion

Chapter 17

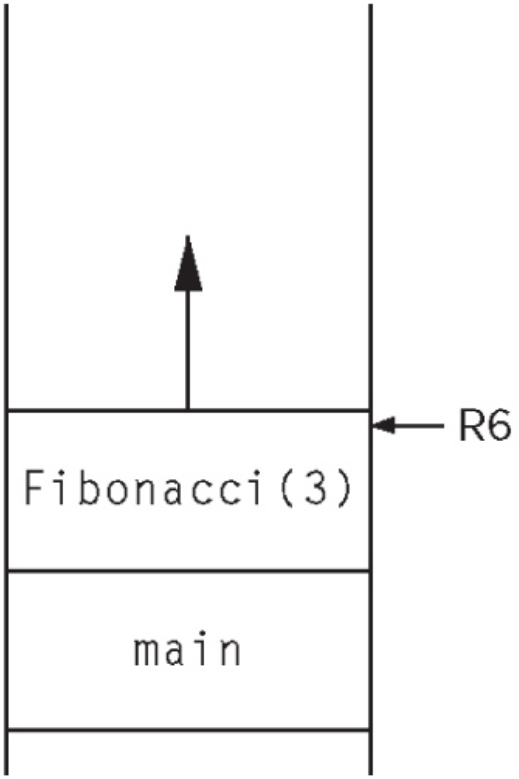
Fibonacci

Compute the *n*-th number in the Fibonacci series.

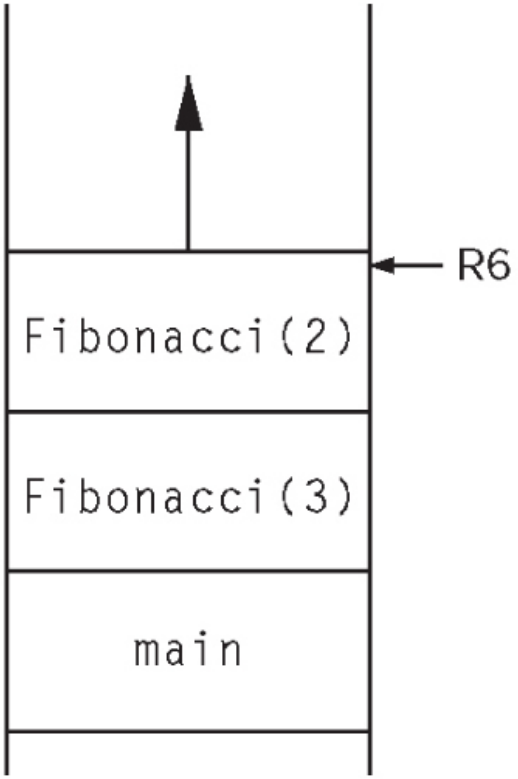
Math: Recurrence Equation	C: Recursive Function

f(*n*) = f(*n* − 1) + f(*n* − 2)
f(1) = 1
f(0) = 1
	`int Fibonacci(int n) {`
	`  int sum;`
	`  if (n == 1 || n == 0)`
	`  return 1;`
	`  else`
	`  sum = Fibonacci(n-1) +`
	`  Fibonacci(n-2);`
	`  return sum;`
	`}`

Fibonacci: Calling Sequence 1



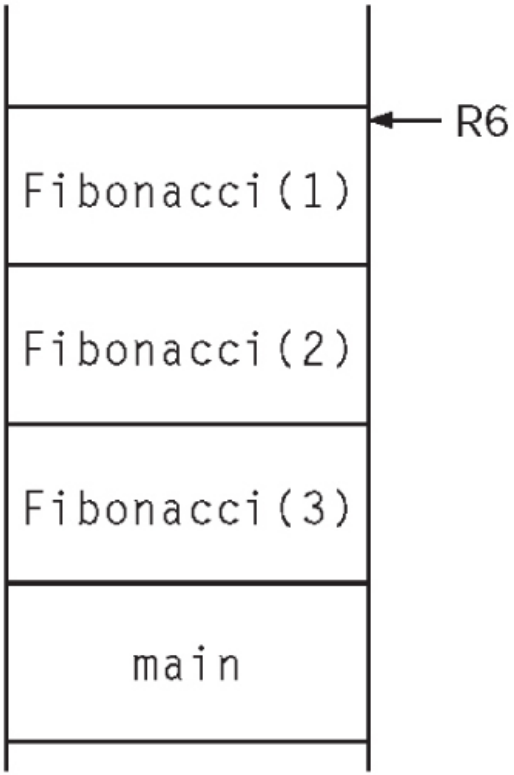
Step 1: Initial call



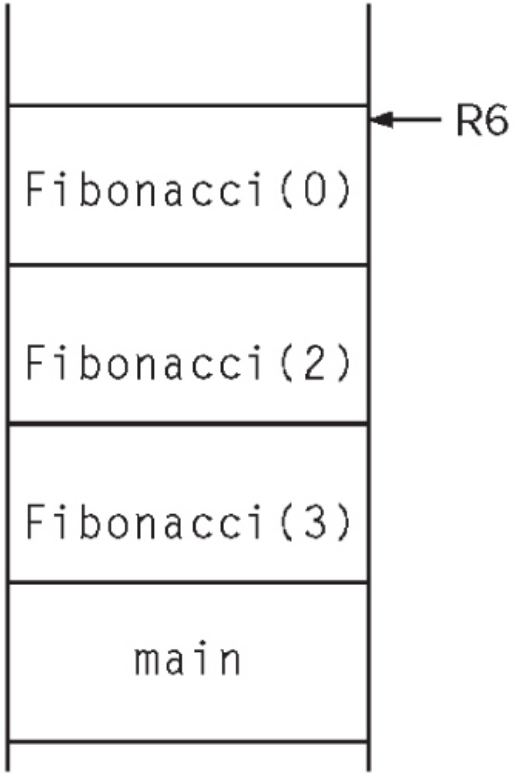
Step 2: Fibonacci(3) calls Fibonacci(2)

Access the text alternative for slide images.

Fibonacci: Calling Sequence 2



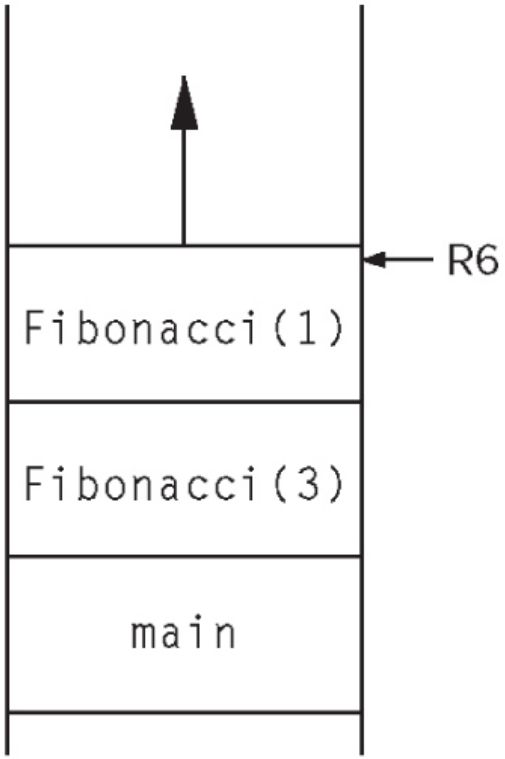
Step 3: Fibonacci(2) calls Fibonacci(1)



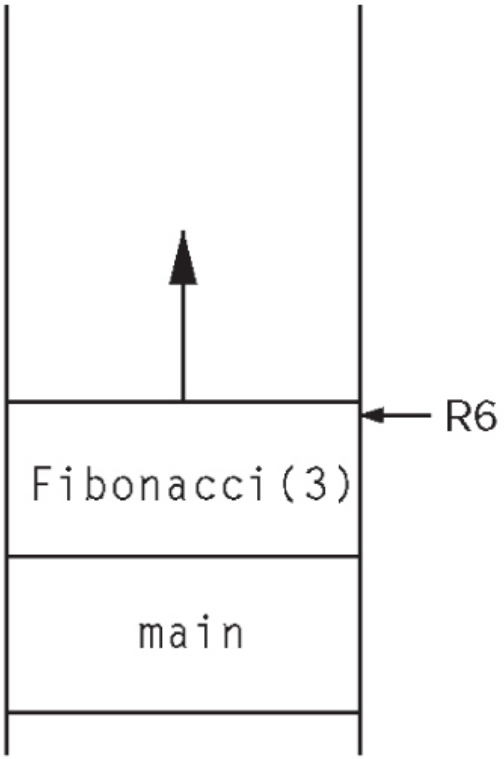
Step 4: Fibonacci(2) calls Fibonacci(0)

Access the text alternative for slide images.

Fibonacci: Calling Sequence 3



Step 5: Fibonacci(3) calls Fibonacci(1)



Step 6: Back to the starting point

Access the text alternative for slide images.

Translate `Fibonacci` to LC-3 1

```
Fibonacci
   ADD R6, R6, #-1    ; space for return val
   ADD R6, R6, #-1    ; push return addr
   STR R7, R6, #0
   ADD R6, R6, #-1    ; push dynamic link
   STR R5, R6, #0
   ADD R5, R6, #-1    ; set frame pointer
   ADD R6, R6, #-2    ; push local (sum) and one temp variable

   ;;; if (n == 0 || n == 1)

   LDR R0, R5, #4     ; get n
   BRn FIB_ELSE       ; skip to recursive case
   ADD R0, R0, #-1
   BRp FIB_ELSE       ; skip to recursive case

   ;;; return 1

   AND R0, R0, #0
   ADD R0, R0, #1
   STR R0, R5, #3     ; store to return value
   BRnzp FIB_END      ; finish return
```

Translate `Fibonacci` to LC-3 2

```
;;; else sum = Fibonacci(n-1) + Fibonacci(n-2)

FIB_ELSE
  LDR R0, R5, #4     ; push n-1
  ADD R0, R0, #-1
  ADD R6, R6, #-1
  STR R0, R6, #0
  JSR Fibonacci      ; Fibonacci(n-1)
  LDR R0, R6, #0     ; get return value
  ADD R6, R6, #2     ; pop r.v. and args
  STR R0, R5, #-1    ; store to temp variable

  LDR R0, R5, #4     ; push n-2
  ADD R0, R0, #-2
  ADD R6, R6, #-1
  STR R0, R6, #0
  JSR Fibonacci      ; Fibonacci(n-2)
  LDR R0, R6, #0     ; get return val and pop stack
  ADD R6, R6, #2
  LDR R1, R5, #-1    ; get temp, compute sum
  ADD R0, R0, R1
  STR R0, R5, #0
```

Translate `Fibonacci` to LC-3 3

```
    ;;; return sum
    LDR R0, R5, #0  ; store to return value
    STR R0, R5, #3
FIB_END
    ADD R6, R6, #2  ; pop local/temp
    LDR R5, R6, #0  ; pop dynamic link
    ADD R6, R6, #1
    LDR R7, R6, #0  ; pop return address
    ADD R6, R6, #1
    RET
```

Fibonacci Observations

Generated code uses stack to store local versions of `sum`.

No additional code is needed to support recursion -- it comes as a natural consequence of the way functions are implemented.

- Why is it important that we push the return address (R7) in the preamble?

Compiler creates a temporary variable to hold the value of `Fibonacci(n-1)` while it computes `Fibonacci(n-2)`. This is a common technique for saving temporary state while evaluating a complex expression -- in this case, the sum of two function calls.

- Why can't we just keep the value in R0 and use R1 for the second call?

As noted in Chapter 8, this is easy to write but very inefficient, because it recomputes the same value again and again.

Because learning changes everything.®

www.mheducation.com