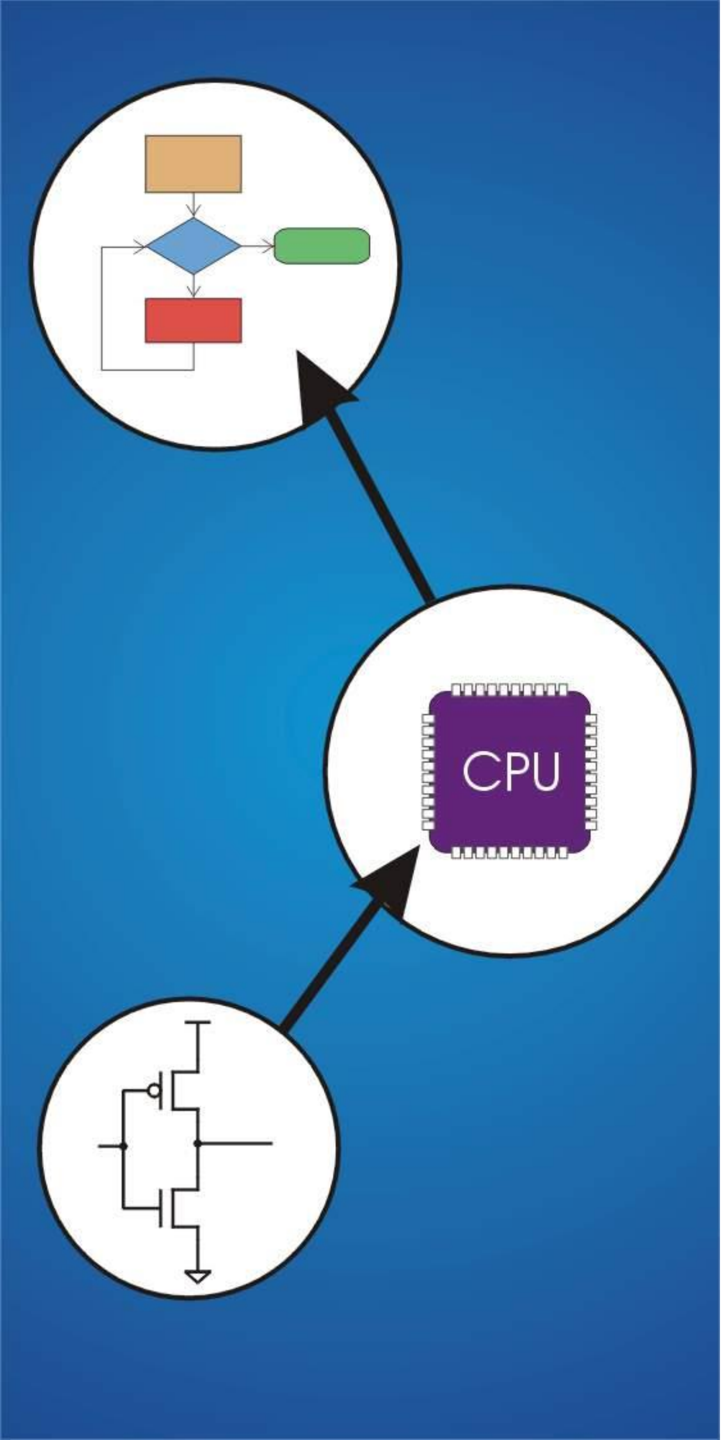


Chapter 17
Recursion

What is Recursion?

A recursive function is one that solves its task by calling itself on smaller pieces of data.

- **Similar to recurrence function in mathematics.**
- **Like iteration -- can be used interchangeably; sometimes recursion results in a simpler solution.**

Example: Running sum ($\sum_{1}^{n} i$)

Mathematical Definition:

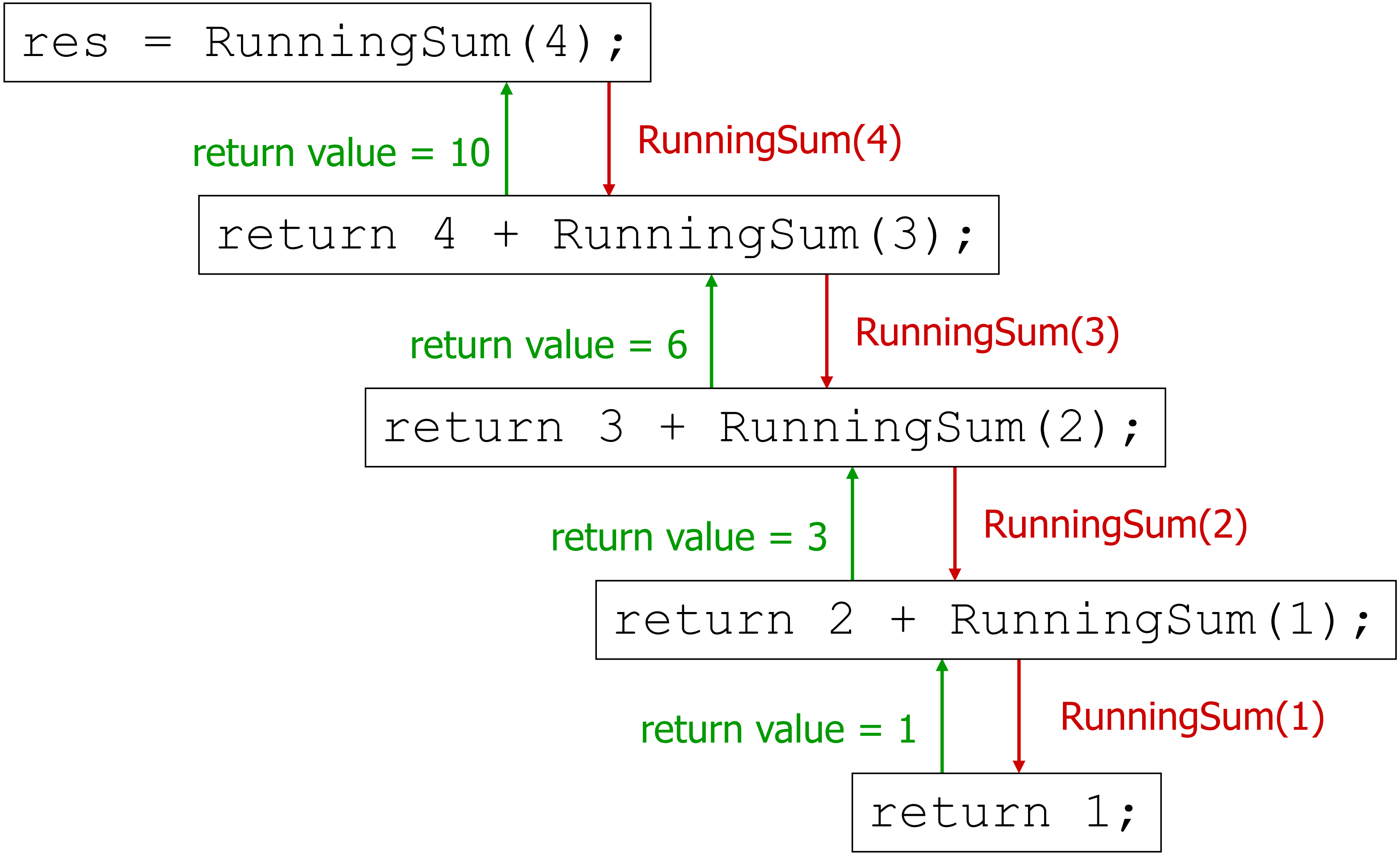
RunningSum(1) = 1

RunningSum(n) =
n + RunningSum(n-1)

Recursive Function:

```
int RunningSum(int n) {
  if (n == 1)
    return 1;
  else
    return n + RunningSum(n-1);
}
```

Executing RunningSum



```
res = RunningSum(4);
```

RunningSum(4)

return value = 10

```
return 4 + RunningSum(3);
```

RunningSum(3)

return value = 6

```
return 3 + RunningSum(2);
```

RunningSum(2)

return value = 3

```
return 2 + RunningSum(1);
```

RunningSum(1)

return value = 1

```
return 1;
```

High-Level Example: Binary Search

Given a sorted set of exams, in alphabetical order, find the exam for a particular student.

1. Look at the exam halfway through the pile.

2. If it matches the name, we're done;
if it does not match, then...

3a. If the name is greater (alphabetically), then
search the upper half of the stack.

3b. If the name is less than the halfway point, then
search the lower half of the stack.

Binary Search: Pseudocode

Pseudocode is a way to describe algorithms without completely coding them in C.

```
FindExam(studentName, start, end)
{
  halfwayPoint = (end + start)/2;
  if (end < start)
    ExamNotFound();  /* exam not in stack */
  else if (studentName == NameOfExam(halfwayPoint))
    ExamFound(halfwayPoint); /* found exam! */
  else if (studentName < NameOfExam(halfwayPoint))
    /* search lower half */
    FindExam(studentName, start, halfwayPoint - 1);
  else /* search upper half */
    FindExam(studentName, halfwayPoint + 1, end);
}
```

Detailed Example: Fibonacci Numbers

Mathematical Definition:

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1$$

$$f(0) = 1$$

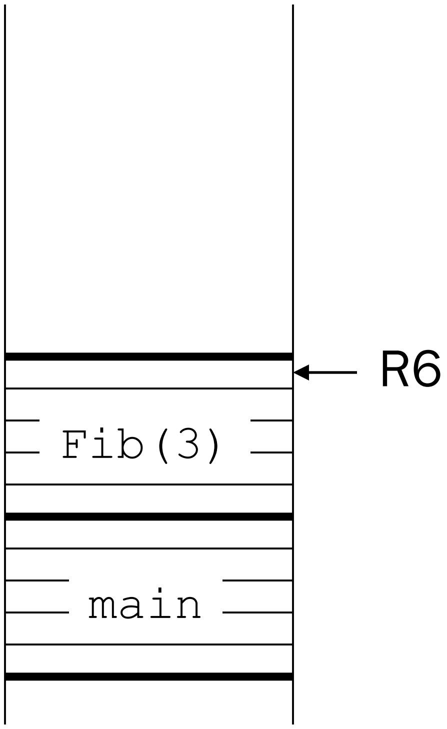
In other words, the n-th Fibonacci number is the sum of the previous two Fibonacci numbers.

Fibonacci: C Code

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

Activation Records

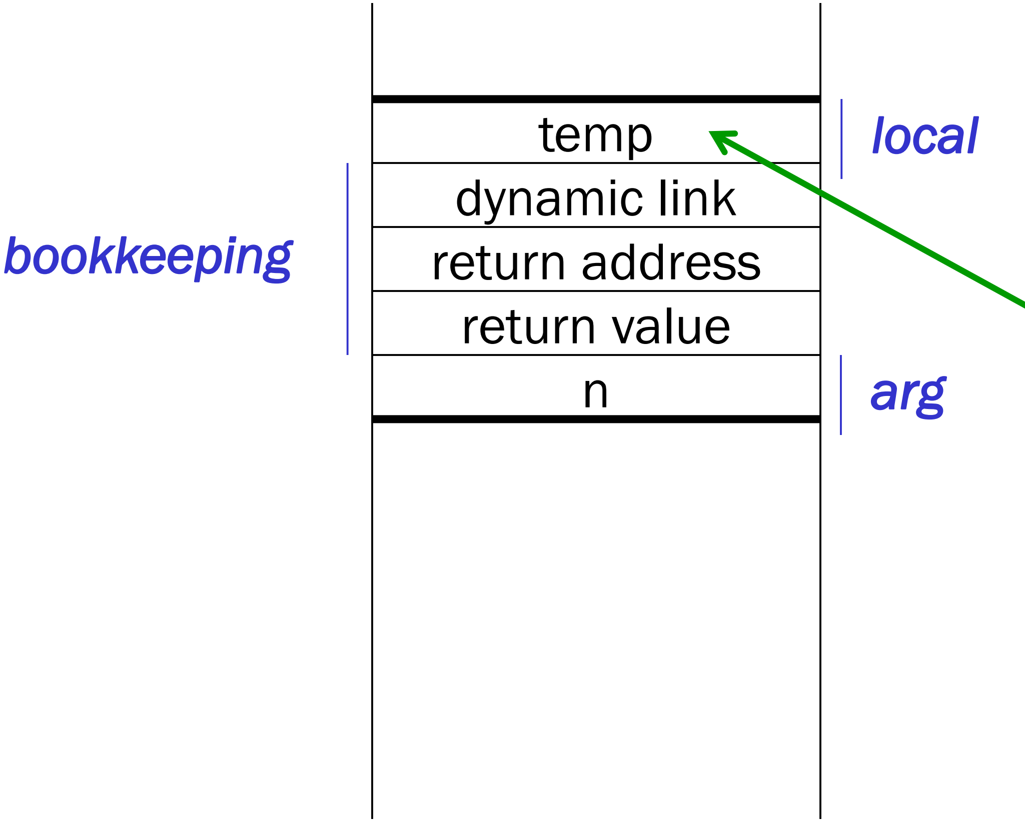
main calls Fibonacci(3)



- **Whenever a function is invoked, a new activation record is pushed onto the stack.**
- **Stack grows from higher to lower addresses.**
- **The stack pointer SP points to the last filled location.**
- **In LC3, R6 serves as the SP.**

Fibonacci: LC-3 Code

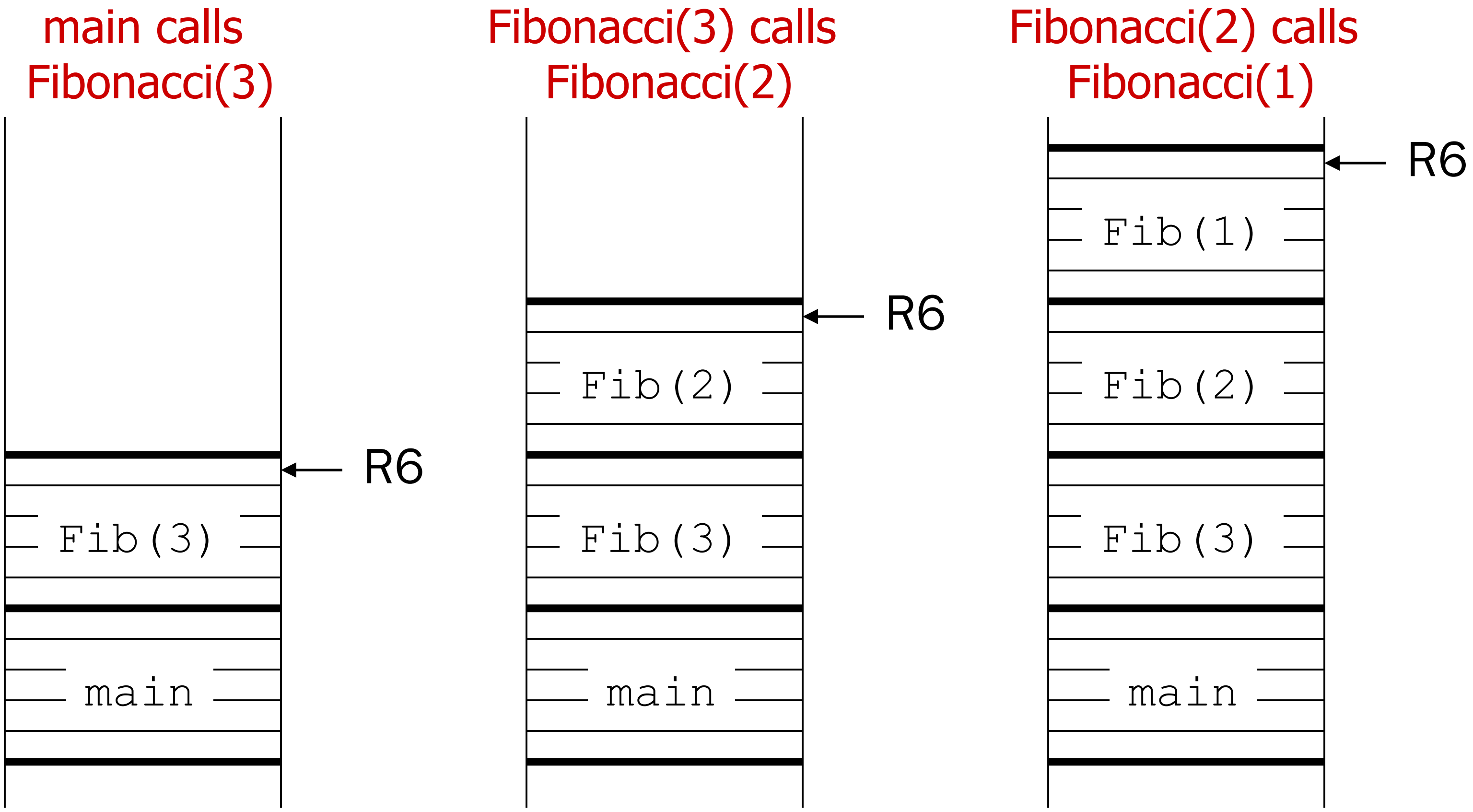
Activation Record



Compiler generates temporary variable to hold result of first Fibonacci call.

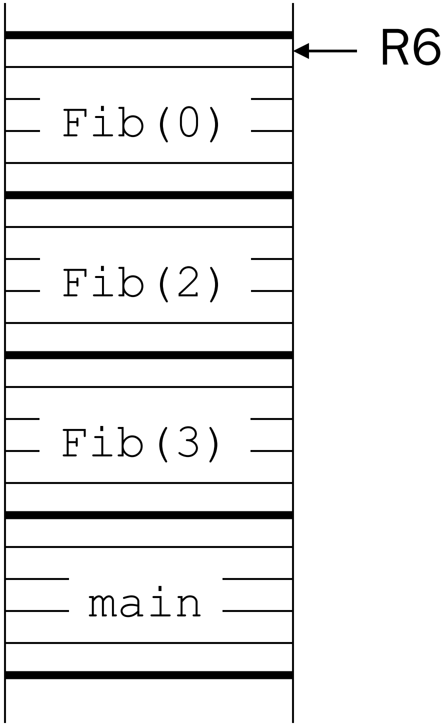
Activation Records

**Whenever Fibonacci is invoked,
a new activation record is pushed onto the stack.**

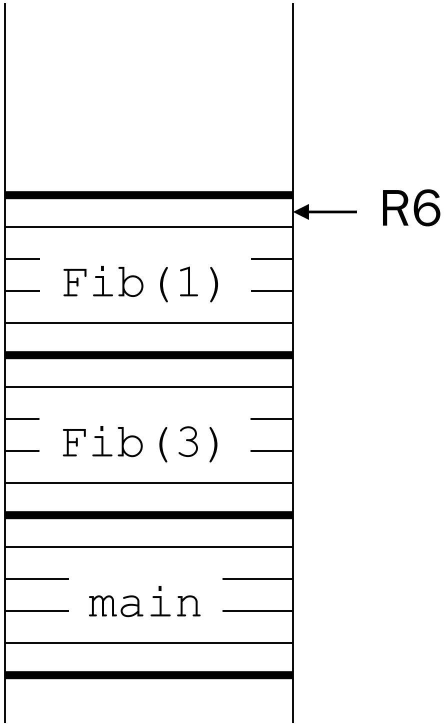


Activation Records (cont.)

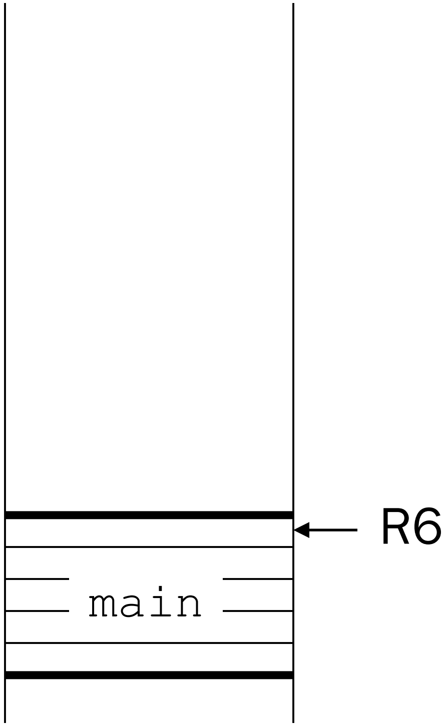
Fibonacci(1) returns,
Fibonacci(2) calls
Fibonacci(0)



Fibonacci(2) returns,
Fibonacci(3) calls
Fibonacci(1)



Fibonacci(3)
returns



Tracing the Function Calls

If we are debugging this program,
we might want to trace all the calls of Fibonacci.

- **Note: A trace will also contain the arguments passed into the function.**

For Fibonacci(3), a trace looks like:

```
Fibonacci(3)
 Fibonacci(2)
  Fibonacci(1)
  Fibonacci(0)
 Fibonacci(1)
```

What would trace of Fibonacci(4) look like?

A Final C Example: Printing an Integer

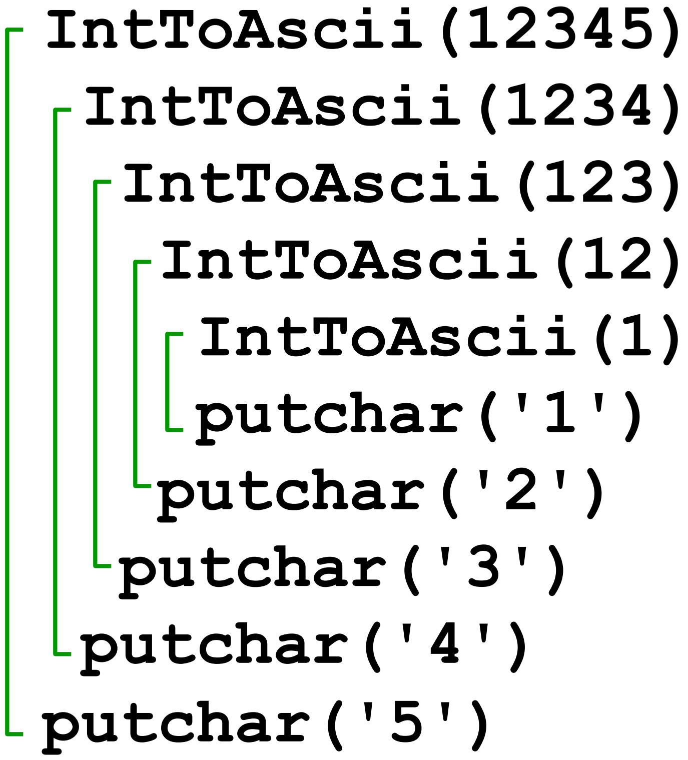
Recursively converts an unsigned integer as a string of ASCII characters.

- **If integer <10, convert to char and print.**
- **Else, call self on first (n-1) digits and then print last digit.**

```
void IntToAscii(int num) {
  int prefix, currDigit;
  if (num < 10)
    putchar(num + '0');  /* prints single char */
  else {
    prefix = num / 10;   /* shift right one digit */
    IntToAscii(prefix);  /* print shifted num */
    /* then print shifted digit */
    currDigit = num % 10;
    putchar(currDigit + '0');
  }
}
```

Trace of IntToAscii

Calling IntToAscii with parameter 12345:



```
IntToAscii(12345)
 IntToAscii(1234)
  IntToAscii(123)
   IntToAscii(12)
    IntToAscii(1)
    putchar('1')
   putchar('2')
  putchar('3')
 putchar('4')
putchar('5')
```

LC-2 Code Skip for now

LC-2 Code (part 1 of 3)

```
Fibonacci  ADD  R6, R6, #-2   ; skip ret val, push ret 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   ; space for locals and temps

           LDR  R0, R5, #4    ; load n
           BRz  FIB_BASE      ; check for terminal cases
           ADD  R0, R0, #-1
           BRz  FIB_BASE
```

LC-3 Code (part 2 of 3)

```
LDR  R0, R5, #4   ; read parameter n
ADD  R0, R0, #-1  ; calculate n-1
ADD  R6, R6, #-1  ; push n-1
STR  R0, R6, #0
JSR  Fibonacci    ; call self

LDR  R0, R6, #0   ; pop return value
ADD  R6, R6, #1
STR  R0, R5, #-1  ; store in temp
LDR  R0, R5, #4   ; read parameter n
ADD  R0, R0, #-2  ; calculate n-2
ADD  R6, R6, #-1  ; push n-2
STR  R0, R6, #0
JSR  Fibonacci    ; call self
```

LC-3 Code (part 3 of 3)

```
           LDR   R0, R6, #0   ; pop return value
           ADD   R6, R6, #1
           LDR   R1, R5, #-1  ; read temp
           ADD   R0, R0, R1   ; Fibonacci(n-1) + Fibonacci(n-2)
           BRnzp  FIB_END     ; all done

FIB_BASE   AND   R0, R0, #0   ; base case – return 1
           ADD   R0, R0, #1

FIB_END    STR   R0, R5, #3   ; write return value (R0)
           ADD   R6, R5, #1   ; pop local variables
           LDR   R5, R6, #0   ; pop dynamic link
           ADD   R6, R6, #1
           LDR   R7, R6, #0   ; pop return address
           ADD   R6, R6, #1
           RET
```