Chapter 17
Recursion

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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 \( \sum_{i=1}^{n} i \)

Mathematical Definition:
\[
\text{RunningSum}(1) = 1 \\
\text{RunningSum}(n) = n + \text{RunningSum}(n-1)
\]

Recursive Function:
\[
\text{int RunningSum(int n)} 
\{
\text{if (n == 1) return 1}; \\
\text{else return n + RunningSum(n-1)};
\}
\]

What is Recursion?

### Executing RunningSum

```
res = RunningSum(4);
res = RunningSum(3);
res = RunningSum(2);
res = RunningSum(1);
return value = 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.

```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

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

Activation Records

Whenever Fibonacci is invoked, a new activation record is pushed onto the stack.
### Activation Records (cont.)

- Fibonacci(1) returns, Fibonacci(0) calls Fibonacci(0)
- Fibonacci(2) returns, Fibonacci(3) calls Fibonacci(0)
- Fibonacci(3) returns, Fibonacci(3) calls Fibonacci(0)

### 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)
- What would trace of Fibonacci(4) look like?

### Fibonacci: LC-3 Code

#### Activation Record

- temp
- dynamic link
- return address
- return value
- n

#### Local

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

### 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.

```c
void IntToAscii(int num)
{
    int prefix, currDigit;
    if (num < 10)
        putchar(num + '0'); /* print number */
    else {
        prefix = num / 10; /* previous digits */
        digit = num % 10; /* current digit */
        IntToAscii(prefix); /* recursive call */
        putchar(digit + '0'); /* print digit */
    }
}
```
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')
```