Computer Programming 143 – Lecture 13
Functions IV

Electrical and Electronic Engineering Department
University of Stellenbosch

Corné van Daalen
Thinus Booysen
Willie Smit
Willem Jordaan

Module Overview

- Chap 1: Introduction to Computer Systems
- Chap 2: Introduction to C
- Chap 3: Structured Program Development
- Chap 9: Formatted Input/Output
- Chap 4: Program Control
- Chap 5: Functions

- Chap 6: Arrays
- Chap 7: Pointers
- Chap 10: Structures
- Chap 11: File Processing
- Chap 12: Data Structures
- Chap 16: Object-Oriented Programming

Lecture Overview

1. 5.14 Recursion
2. 5.15 Example: Fibonacci Series
3. 5.16 Recursion vs. Iteration

5.14 Recursion I

Function calls so far

- Functions call one another in a disciplined and hierarchical manner
- Functions only call other functions – not themselves

Definition of recursion

A **recursive function** is a function that calls itself, either directly or indirectly through another function

5.14 Recursion II

Recursive problem solving

- Function can only solve simplest case(s) directly (**base case(s)**):
  - Function called with base case: returns result of base case
- Function called with more complex case:
  - Breaks problem into two pieces: one it “knows how to do” and one it “does not know how to do”
  - The second part must be a simpler or smaller version of the original problem
  - To solve the second part, the function calls itself (**recursion step**)
  - Function stays “open” while recursion step executes

5.14 Recursion III

Recursion analogy

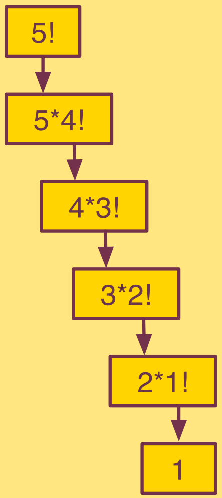
5.14 Recursion IV

Factorial 5!

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ &= 5 \times (4 \times 3 \times 2 \times 1) \\ &= 5 \times (4!) \end{aligned}$$

Similarly,

$$\begin{aligned} 4! &= 4 \times (3!) \\ 3! &= 3 \times (2!) \\ 2! &= 2 \times (1!) \\ 1! &= 1 \end{aligned}$$



5.14 Recursion V

Factorial n!

$$n! = \begin{cases} 1, & n = 1 \\ n \times (n-1)!, & n > 1 \end{cases}$$

Problem

- Write a function to calculate the factorial of a number recursively.

Pseudocode

function: factorial of integer n

if n is less than or equal to 1
set the variable to return to 1

else calculate the factorial of (n-1), i.e. call factorial (n-1)
set the variable to return to n multiplied by the factorial of (n-1)

5.14 Recursion VI

Factorial: recursive implementation

```
long factorial( long n )
{
   long x;
   // base case
   if ( n <= 1 )
      x = 1;
   else    // recursive step
      x = n * factorial( n - 1 );
   return x;
} // end function factorial
```

Refer to Fig. 5.18 in Deitel & Deitel for full program listing

5.15 Example: Fibonacci Series I

Fibonacci series

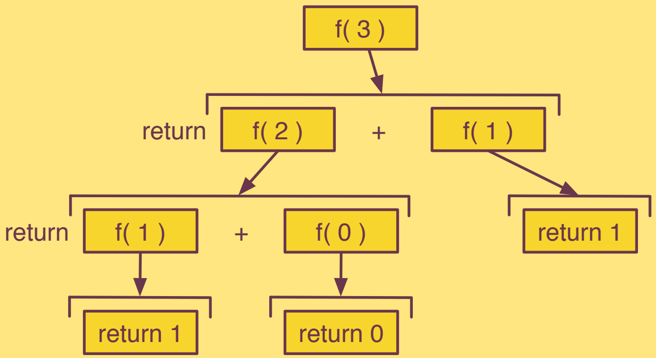
The Fibonacci series, 0, 1, 1, 2, 3, 5, 8, 13, 21, . . ., begins with 0 and 1 and has the property that each subsequent Fibonacci number is the sum of the previous two Fibonacci numbers

Any Fibonacci number can therefore be calculated as

$$\text{Fibonacci}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2), & n > 1 \end{cases}$$

5.15 Example: Fibonacci Series II

Fibonacci series (cont'd. . .)



5.15 Example: Fibonacci Series III

Problem

- Write a function to calculate the Fibonacci series recursively.

Pseudocode

function: fibonacci of integer n
if n is 0 or 1
set return variable to n

else
calculate the fibonacci of (n-1), i.e. call fibonacci (n-1)
calculate the fibonacci of (n-2), i.e. call fibonacci (n-2)
set return variable to the sum of fibonacci(n-1) and fibonacci(n-2)

5.15 Example: Fibonacci Series IV

Fibonacci series: recursive implementation

```
long fibonacci( long n )
{
   long x;
   // base case
   if ( n == 0 || n == 1 ) {
      x = n;
   } // end if
   else { // recursive step
      x = fibonacci( n - 1 ) + fibonacci( n - 2 );
   } // end else
   return x;
} // end function fibonacci
```

Refer to Fig. 5.19 in Deitel & Deitel for full program listing

5.16 Recursion vs. Iteration I

Recursion vs. Iteration

- All problems that can be solved with recursion can also be solved with iteration
- Recursion is more processor and memory intensive
- Sometimes the recursion solutions are more elegant

Perspective

Today

Functions IV

- Definition of recursion
- Example: Fibonacci series
- Recursion vs. iteration

Next lecture

Arrays I

- Introduction to, definition of and use of arrays

Homework

1. Study Sections 5.14-5.16 in Deitel & Deitel
2. Do Self Review Exercises 5.1(k)-(q) in Deitel & Deitel
3. Do Exercises 5.34, 5.36 in Deitel & Deitel