Recursion

Lecture 11

Menu

- recursive functions
- factorial function
- fibonacci function
- recursion vs iteration

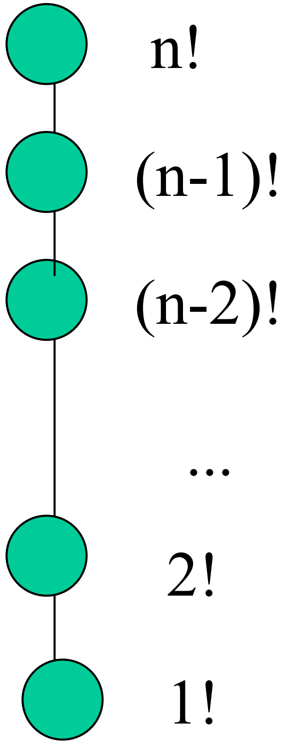
recursive functions

- A recursive function is a function that calls itself directly or indirectly
- Related to recursive problem solving: binary tree traversal (**divide & conquer**)
- The function knows how to solve a base case (**stopping rule**)
- A recursion step is needed to divide the problem into sub-problems (**key step**)
- Need to check for **termination**

factorial

- $1! = 1$
- $5! = 5 * 4 * 3 * 2 * 1$ or
- $5! = 5 * 4!$

- A recursive definition:
- $n! = n * (n-1)!$



recursion tree for n!

factorial function fac

- base case:

  ```
  if (number <= 1) return 1;
  ```

- recursive step:

  ```
  return ( number * fac (number - 1));
  ```

```
#include <stdio.h>
long fac(long);

main()
{
  int i;
 for (i=1; i <=5; i++)
      printf("%d! = %ld\n", i, fac(i));
 return 0;
}
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

```
long fac(long number)
{
if (number <= 1)
   return 1;
else
 return ( number * fac (number - 1));
}
```

How does fac work?

```
long fac(long number)
{
if (number <= 1)
    return 1;
else
 return ( number * fac (number - 1));
}

fac(1) --> if (1 <= 1) return 1; --> 1
fac(2) --> else return ( 2 * fac (2 - 1));
           --> return (2 * fac(1));
           --> return (2* 1);
```

```
long fac(long number)
{
if (number <= 1)
    return 1;
else
 return ( number * fac (number - 1));
}

fac(3) --> else return ( 3 * fac (3 - 1));
          --> return ( 3  * fac (2));
          --> return (3* (2 * fac(1)) );
          --> return (3* 2*1);
```

fibonacci function

- fibonacci series: 0,1,1,2,3,5,8,13,21, ...
- **base case:**
- fibonacci (0) = 0
- fibonacci (1) = 1
- **recursive step:**
- fibonacci (n) = fibonacci (n-1) + fibonacci (n-2)

```
/* Recursive definition of function fibonacci */

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

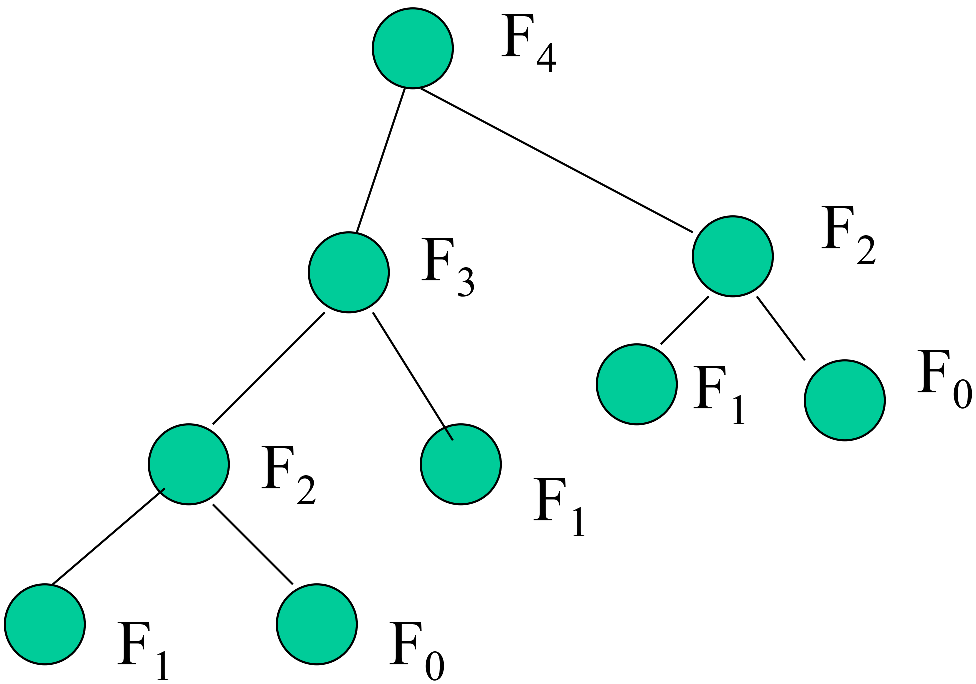
```
#include <stdio.h>

long fibonacci(long);

main()
{
  long result, number;

  printf("Enter an integer: ");
  scanf("%ld", &number);
  result = fibonacci(number);
  printf("Fibonacci(%ld) = %ld\n", number, result);
  return 0;
}
```

```
Enter an integer: 0
Fibonacci(0) = 0
Enter an integer: 1
Fibonacci(1) = 1
Enter an integer: 2
Fibonacci(2) = 1
Enter an integer: 3
Fibonacci(3) = 2
Enter an integer: 4
Fibonacci(4) = 3
```

How does fibonacci work?

```
fibonacci (0) --> if (0 == 0 || 0 == 1) return 0;
fibonacci (1) --> if (1 == 0 || 1 == 1) return 1;
fibonacci (2) --> else return fibonacci(2 - 1) + fibonacci(2 - 2);
                  --> fibonacci(1) + fibonacci(0);
                  --> 1  + 0
fibonacci (3) --> else return fibonacci(3 - 1) + fibonacci(3 - 2);
                  --> return fibonacci(2) + fibonacci(1);
fibonacci (4) --> else return fibonacci(4 - 1) + fibonacci(4 - 2);
                  --> return fibonacci(3) + fibonacci(2);
                  -->{ fibonacci(3 - 1) + fibonacci(3 - 2) }+
                     {fibonacci(2 - 1) + fibonacci(2 - 2) }+
                  --> ...
```

Recursion tree for fibonacci(4)



Recursion vs iteration

- iteration: while, for
- recursion uses a selection structure & function calls;
- iteration uses an iterative structure
- recursion & iteration each involves a termination test:
  - recursion ends when a base case recognized
  - iteration ends when the continuation condition fails
- every recursive function can be rewritten as an iterative function
- iteration: efficient but not easy to design
- recursion: slow (cost memory & processor power) but elegant

EXERCISE

- Design a recursive function to solve the following problem: sum up a given array a[0]..a[m-1] & return the sum to the caller

- int sum_up(a, n) // n: number of array elements to sum up

Q&A

- What is the key step in designing a recursive function?
- Every recursive function can be rewritten as an iterative function. (T or F)

Summary

- recursive functions
- factorial function
- fibonacci function
- recursion vs iteration

Readings

- [Mar07] Read 1.3
- [Mar13] Read 1.3