

COMS 1003: Introduction to Computer Programming in C

Recursion

October 4th 2005

Announcements

- Use the blog
- HW3 and 4 released
- mid-semester reviews
 - me, the class, TAs

Outline

- Review
- Recursion

Recursion

- Doing the same work on smaller instances of a problem
- Circular Definitions

Recursive Procedure

- Base cases
 - condition specifying when to stop recursion
- Recursive Step
 - do some small part of the problem
 - invoke same function on reduced problem

Self-Calling Functions

- Entirely legal, but must handle with care

```
//what do I do?  
int a(int x)  
{  
    return a(x);  
}
```

Self-Calling Functions

```
//what do I do?
//what do I do? int a(int x)
int a(int x) {
{           return a(x);
    return a(x); }
}
```



Self-Calling Functions

```
//what do I do?
//what do I do? int a(int x)
int a(int x) {
{
    return a(x);
}
    return a(x); }
```

```
//what do I do?
int a(int x)
{
    return a(x);
}
```

Self-Calling Functions

```
//what do I do?  
int a(int x)  
{  
    return a(x);  
}  
//what do I do?  
int a(int x)  
{  
    return a(x);  
}  
//what do I do?  
int a(int x)  
{  
    return a(x);  
}
```

The diagram illustrates the recursive calls of a self-calling function 'a'. It shows three instances of the function definition, each with a comment '//what do I do?'. Arrows indicate the flow of recursive calls: from the first instance to the second, and from the second to the third. Each arrow starts at a dot on the 'return a(x);' line of one instance and points to the 'int a(int x)' line of the next instance. The third instance does not have an outgoing arrow, representing the base case of the recursion.

Self-Calling Functions

```
//what do I do?  
int a(int x)  
{  
    return a(x);  
}  
//what do I do?  
int a(int x)  
{  
    return a(x);  
}  
//what do I do?  
int a(int x)  
{  
    return a(x);  
}  
//what do I do?  
int a(int x)  
{  
    return a(x);  
}
```

The diagram illustrates the recursive calls in a self-calling function. It shows four instances of the function `int a(int x)` arranged in a descending staircase pattern. Each instance has a line `return a(x);` and a line `int a(int x)`. Arrows indicate the flow of recursive calls: from the `return a(x);` line of one instance to the `int a(int x)` line of the next instance below it. The top instance starts with a comment `//what do I do?` and an arrow pointing to its `int a(int x)` line. The bottom instance also starts with `//what do I do?` and has an arrow pointing to its `int a(x);` line. The arrows show a chain of calls where each function calls itself, eventually leading to a base case (though not explicitly shown here).

Self-Calling Functions

The diagram illustrates the recursive calls of a self-calling function `a`. It features multiple instances of the function definition, with arrows indicating the sequence of calls. Each call is represented by a dot (the call site) and an arrow pointing to the function definition (the callee). The function definition is as follows:

```
//what do I do?  
int a(int x)  
{  
    return a(x);  
}
```

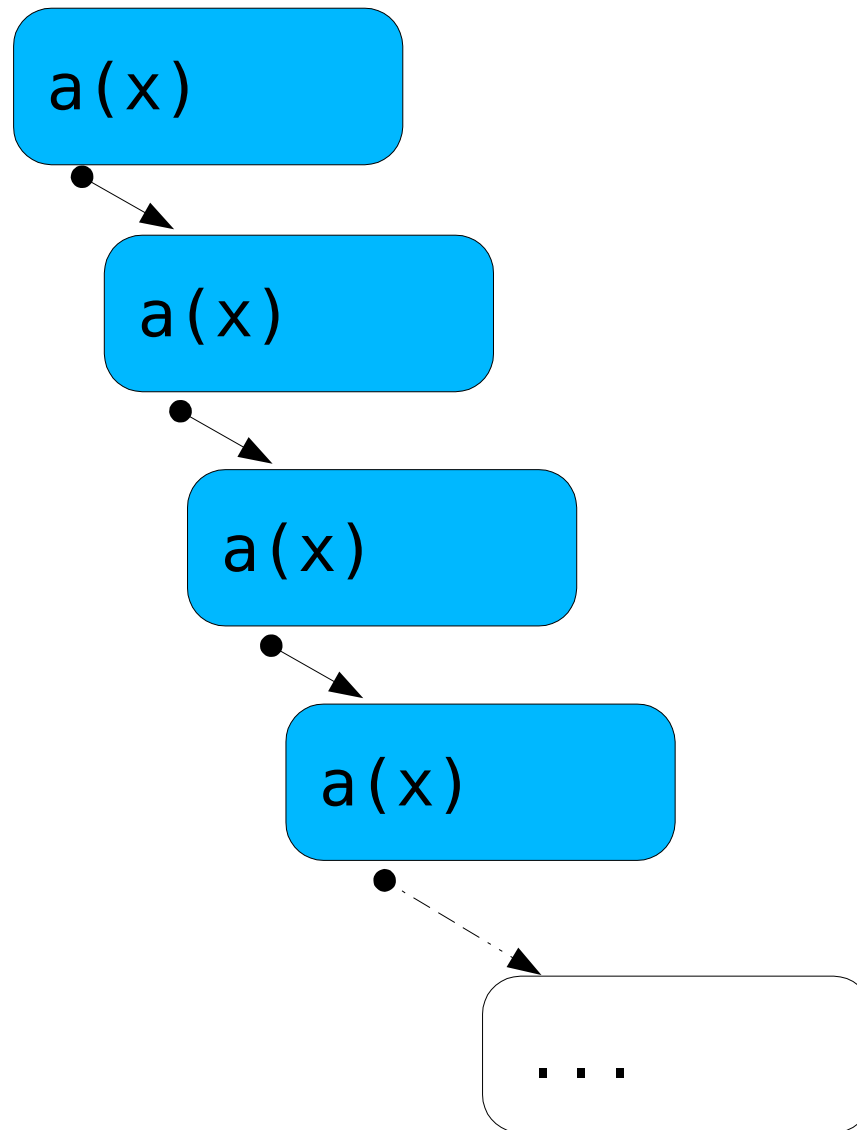
The diagram shows the following call sequence:

- Initial call: A dot at the top left points to the first function definition.
- First recursive call: A dot inside the first function's body points to a second function definition.
- Second recursive call: A dot inside the second function's body points to a third function definition.
- Third recursive call: A dot inside the third function's body points to a fourth function definition.
- Fourth recursive call: A dot inside the fourth function's body points to a fifth function definition.
- Base case: The fifth function definition is the final one shown, with no further calls indicated.

The function definitions are arranged in a staggered, overlapping manner to show the depth of recursion. The text `//what do I do?` is repeated above each function definition.

Call Graph

```
int a(int x)
{
    return a(x);
}
```



Recursive Example: Factorial

- $N! = N * (N-1)!$

- What if $N \leq 0$?

- E.g.,

$$4! = 4 * 3!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1 * 0!$$

$$0! = 1$$

Recursive Example: Factorial

```
/* Function prototype */  
long fact(long);
```

Recursive Example: Factorial

```
/* Function prototype */  
long fact(long);  
  
/* Function definition */  
long fact(long n)  
{  
    //calculate factorial  
    //return answer  
}
```

Recursive Example: Factorial

```
/* Function prototype */  
long fact(long);  
  
/* Function definition */  
long fact(long n)  
{  
    /* recursive step */  
    return n*fact(n-1);  
}
```

Recursive Example: Factorial

```
/* Function prototype */  
long fact(long);  
  
/* Function definition */  
long fact(long n)  
{  
    /* base cases */  
    if(0==n || n==1)  
        return 1;  
    else  
        /* recursive step */  
        return n*fact(n-1);  
}
```