Topic 8
Lecture 8b
Advanced Procedures

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

What's Next (1 of 4)

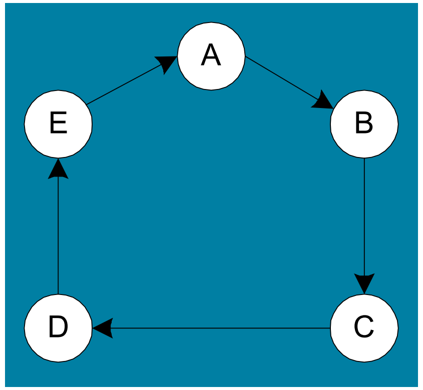
- Stack Frames
- **Recursion**
- Creating Multimodule Programs

Recursion

- What is Recursion?
- Recursively Calculating a Sum
- Calculating a Factorial

What is Recursion?

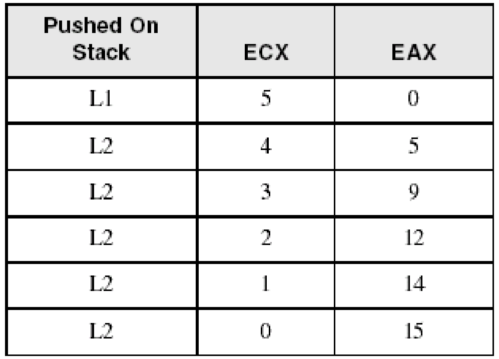
- The process created when . . .
  - A procedure calls itself
  - Procedure A calls procedure B, which in turn calls procedure A
- Using a graph in which each node is a procedure and each edge is a procedure call, recursion forms a cycle:



Recursively Calculating a Sum

The sum procedure recursively calculates the sum of an array of integers. Receives: value on the stack. Returns: EAX = sum

```
sum:
        push ebp                        ; preserve ebp
        mov ebp, esp                    ; start frame
        mov eax, 0                      ; set default return value
        mov ecx, [ebp + 8]              ; move arg into ecx
        cmp ecx, 0                      ; check value
.if:
        je .endif                       ; return default if zero
        dec ecx                         ; decrement value
        push ecx                        ; push as parameter
        call sum                        ; recursive call
        add eax, [ebp + 8]              ; add returned sum to this arg
.endif:
        leave
ret
```

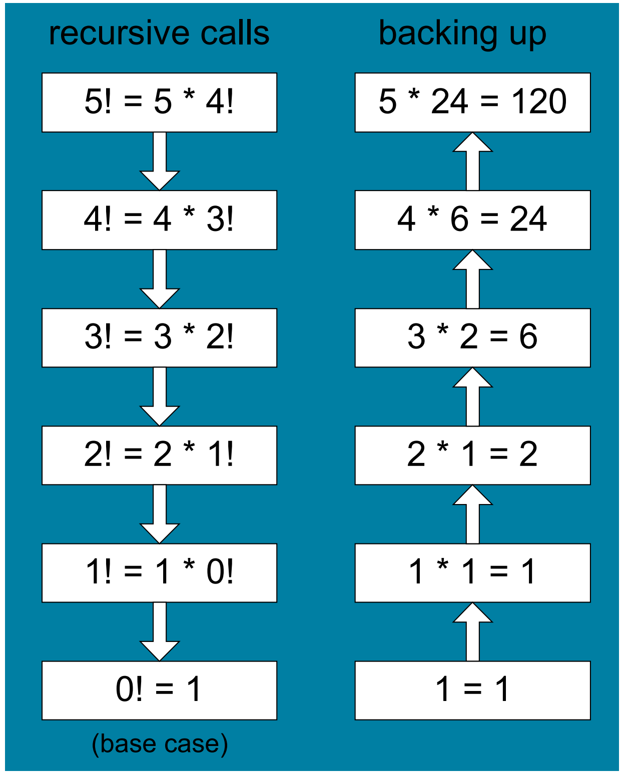


Pushed On Stack	ECX	EAX
L1	5	0
L2	4	5
L2	3	9
L2	2	12
L2	1	14
L2	0	15

Calculating a Factorial (1 of 3)

This function calculates the factorial of integer *n*. A new value of *n* is saved in each stack frame:

```
int function factorial(int n)
{
        if(n == 0)
          return 1;
        else
          return n * factorial(n-1);
}
```



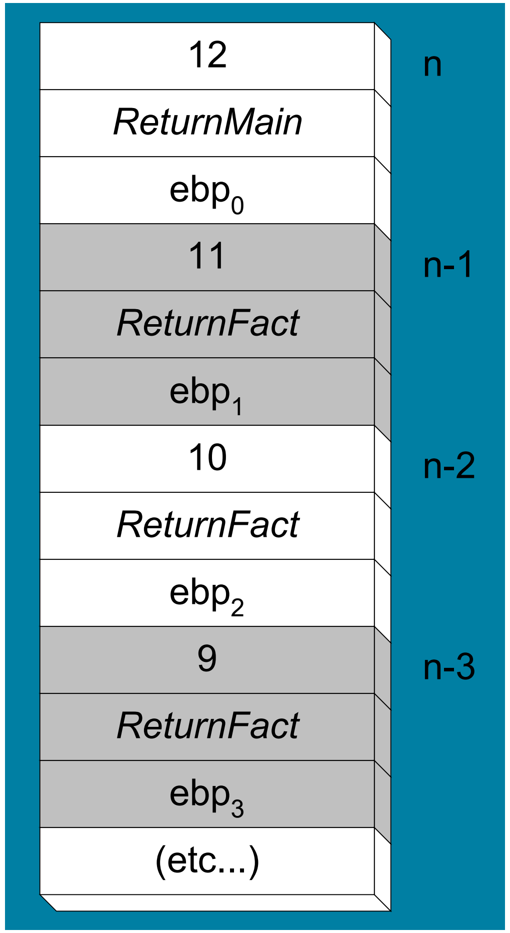
As each call instance returns, the product it returns is multiplied by the previous value of n.

Calculating a Factorial (3 of 3)

Suppose we want to calculate 12!

This diagram shows the first few stack frames created by recursive calls to Factorial

Each recursive call uses 12 bytes of stack space.



12	n
ReturnMain	
ebp_0	
11	n-1
ReturnFact	
ebp_1	
10	n-2
ReturnFact	
ebp_2	
9	n-3
ReturnFact	
ebp_3	
(etc...)	

What's Next (3 of 4)

- Stack Frames
- Recursion
- **Creating Multimodule Programs**

Multimodule Programs

- A multimodule program is a program whose source code has been divided up into separate ASM files.
- Each ASM file (module) is assembled into a separate object file.
- All object files belonging to the same program are linked using the link utility into a single executable file.
    - This process is called static linking

Advantages

- Large programs are easier to write, maintain, and debug when divided into separate source code modules.
- When changing a line of code, only its enclosing module needs to be assembled again. Linking assembled modules requires little time.
- A module can be a container for logically related code and data (think object-oriented here...)
  - encapsulation: procedures and variables are automatically hidden in a module unless you declare them public

Creating a Multimodule Program

- Here are some basic steps to follow when creating a multimodule program:
  - Create the main module
  - Create a separate source code module for each procedure or set of related procedures
  - Create an include file that contains procedure prototypes for external procedures (ones that are called between modules)
  - Use the %include directive to make your procedure prototypes available to each module