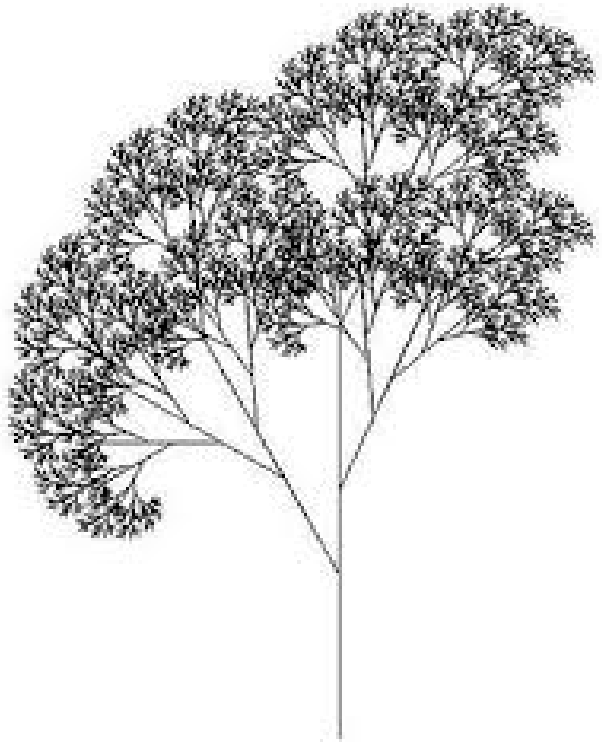
Programming with Data Structures

CS 241-03

Data Structures



Recursion

Geri Lamble

CS 124-03

Reading this Week

Text: Chapter 11

RECURSION BASICS

Recursion

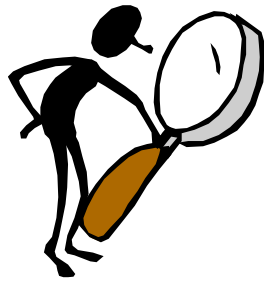
Recursion is defining something in terms of itself.

- Many functions can be defined recursively:
  - Factorial: $n! = n(n-1)!$
  - Differentiation (chain rule): $\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$
  - The binomial coefficient: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$
- We define many data structures recursively
  - A linked list node contains a pointer to a node
  - A binary tree node contains two pointers to nodes
- Euclid's algorithm for GCD is recursive!

Recursive Functions in C++

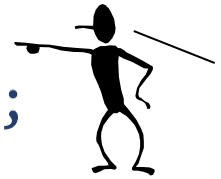
- Most modern programming languages allow recursion in functions;
- In C++, you simply call a function from within itself, e.g.:

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



Recursion– Base case

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



base case

What would happen without that line?

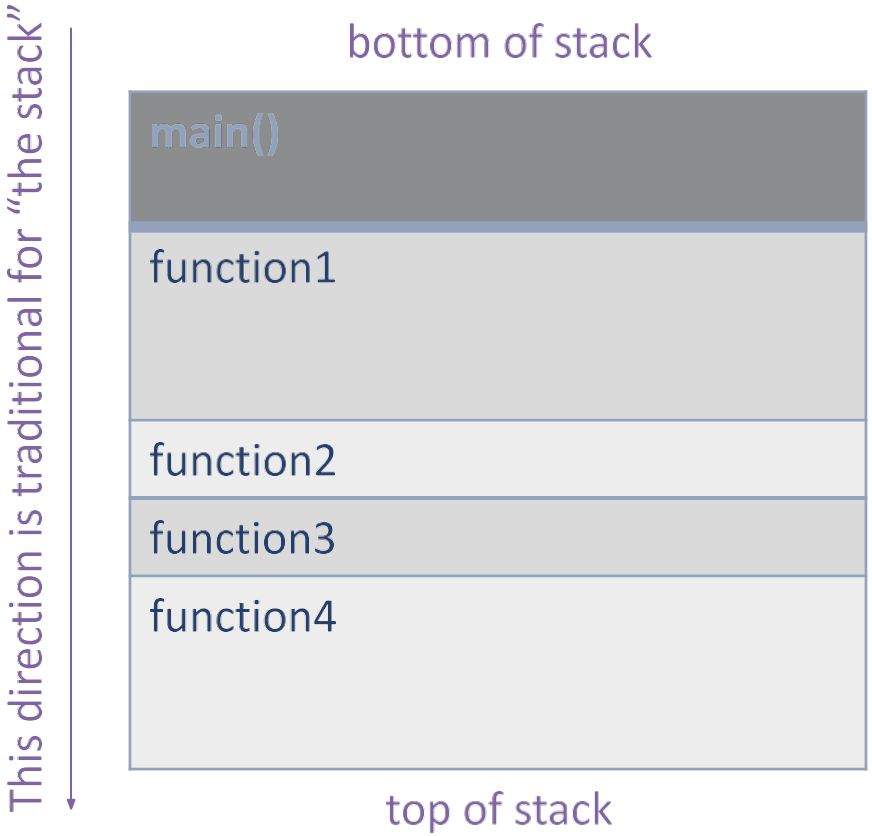
When the input n is 0 we call it the *base case*.

The test for the base case **must** come **before** the recursive call!

RECURSION AND THE STACK

"The Stack" Revisited

When we talk about "the stack", we usually mean a very specific stack; the memory stack of a running program:



Local variables declared in main, return address, other stuff.

Local variables declared in function1; arguments passed by value into function1, return address, other stuff.

Etc.

Each "frame" is created when the function is called, and destroyed when the function exits.

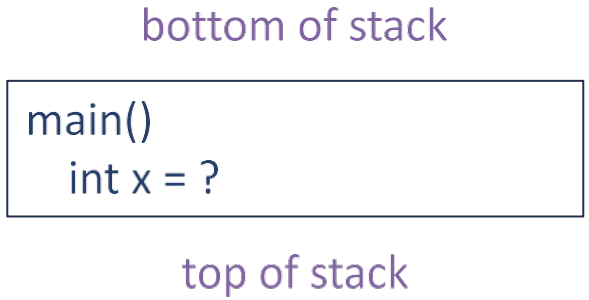
Recursion and the Stack

Key to understanding recursion in C++:

- Each function *call*, not each function, gets an entry on the stack
  - Each stack entry has memory specific to where we are in the recursion – arguments passing *down*
  - Also need to think about values going *up* as we "unwind" the stack

Example: Factorial Start

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



Factorial First Call
factorial(4)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```

main()
  int x = ?

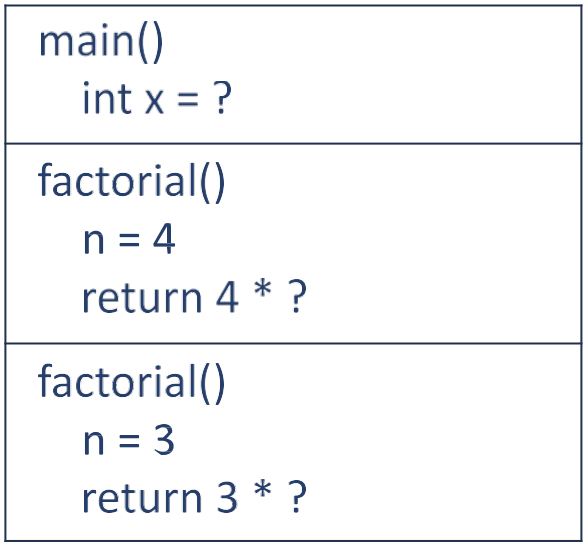
factorial()
  n = 4
  return 4 * ?

top of stack

Factorial Second Call

factorial(3)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```

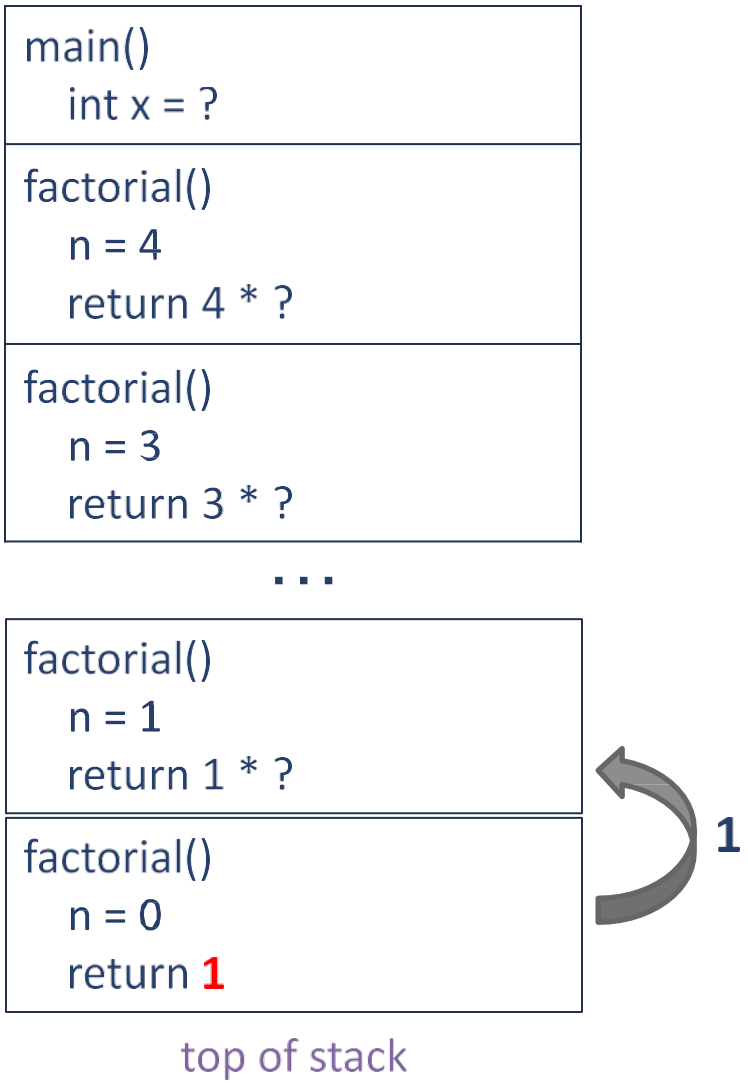


top of stack

Factorial Fifth Call

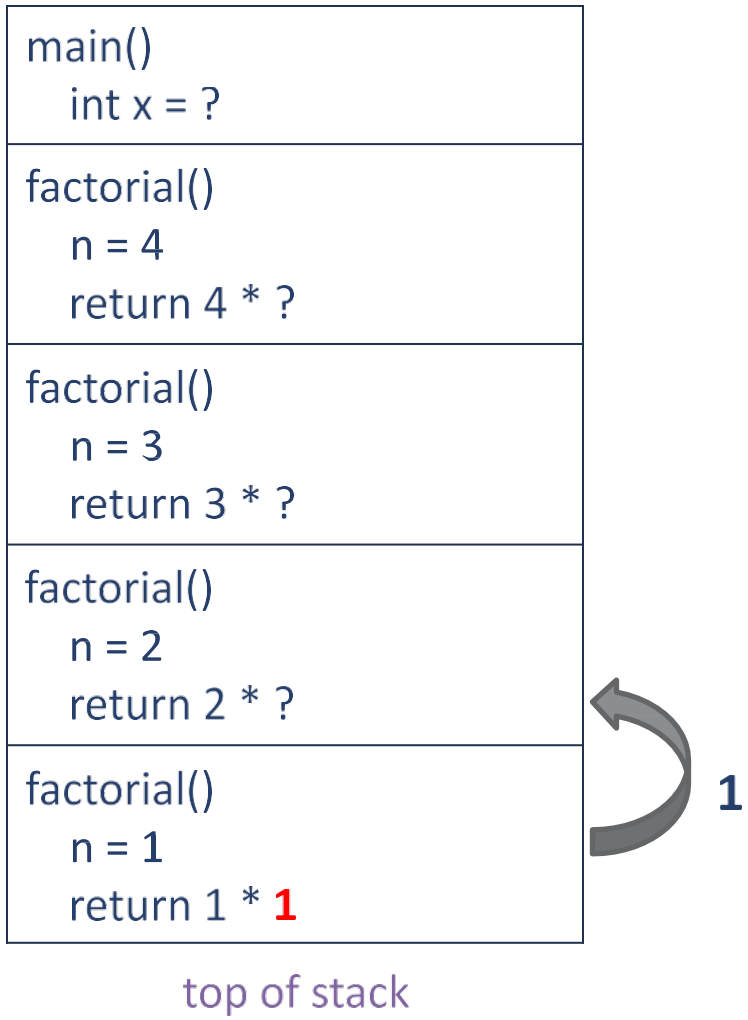
factorial(0) – Base case

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



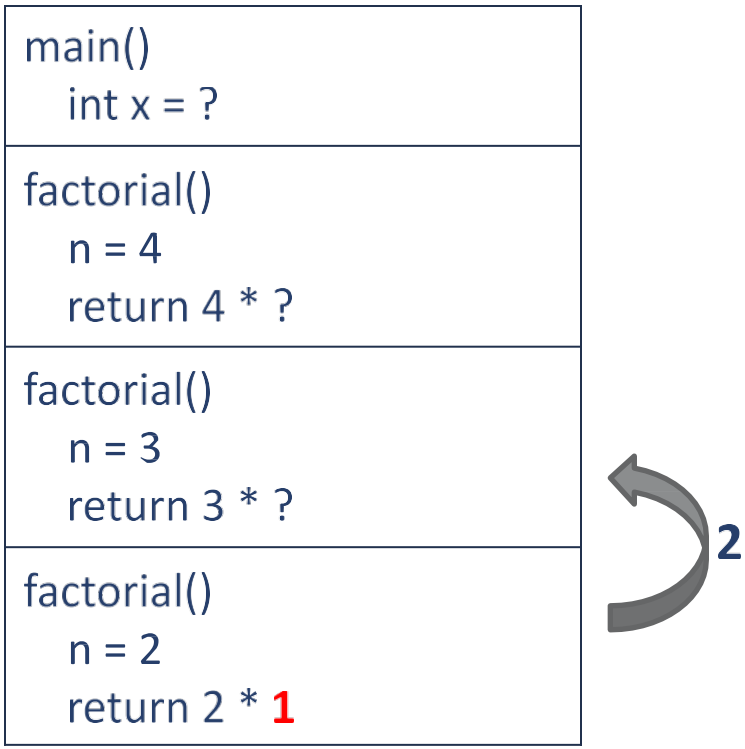
Factorial Unwinding: factorial(1)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



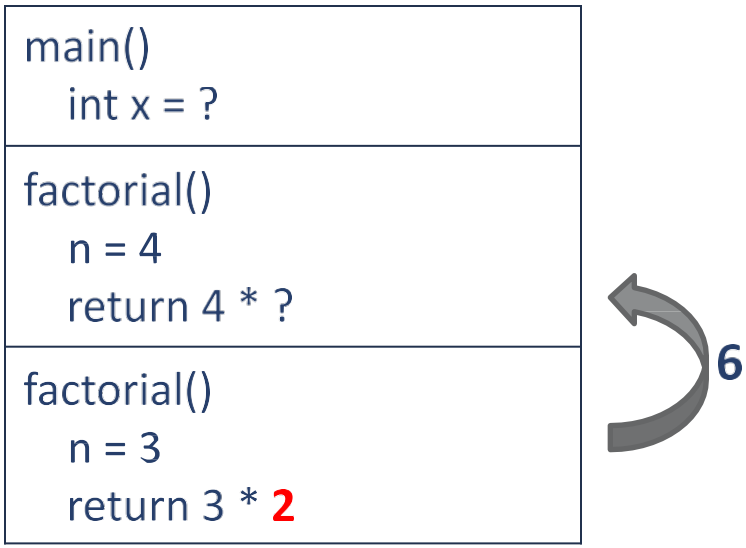
Factorial Unwinding: factorial(2)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



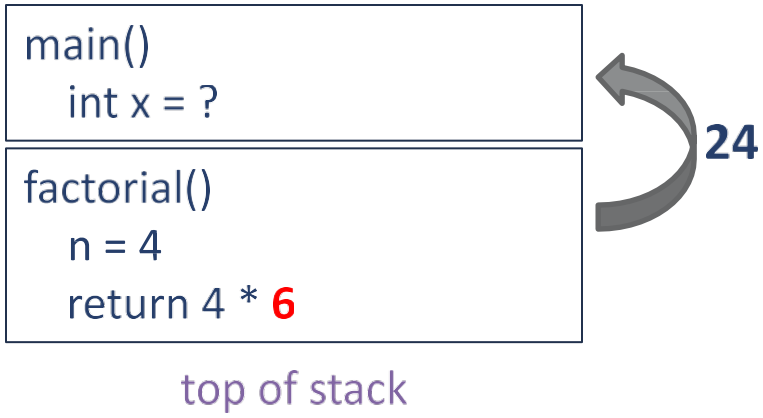
Factorial Unwinding: factorial(3)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



Factorial Unwinding: factorial(4)

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```



Factorial Unwinding: main()

```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(4);
}
```

main()
  int x = **24**

top of stack

MORE RECURSIVE EXAMPLES

Example: Palindrome

- A palindrome is a recursive object; it is:
  - Empty, or
  - A single character, or
  - A palindrome between two of the same character

Base cases

kayak
civic
redivider
etc.

- Here's a recursive test function:

Sometimes need a recursive *helper* function

```
bool is_palindrome(const string &s, int start, int end) {
    if (end <= start) return true;

    return (s[start] == s[end] && is_palindrome(s, start+1, end-1));
}

bool is_palindrome(const string &s) {
    return is_palindrome(s, 0, s.length() - 1);
}
```

Example: Binomial Coefficient

```
unsigned int nchoosek(unsigned int n, unsigned int k) {
   assert(n >= k);
   if (k == 0 || k == n) return 1;

   return nchoosek(n-1, k) + nchoosek(n-1, k-1);
}
```

Note - more than one base case!

Note - two recursive calls!

Common Mistakes

- No base case:

```
void infinite(int n) {
   cout << n << endl;
   infinite(n-1);
}
```

- Recursion step doesn't reduce problem:

```
void infinite2(int n) {
   if (n < 0) return;
   cout << n << endl;
   infinite2(n);
}
```

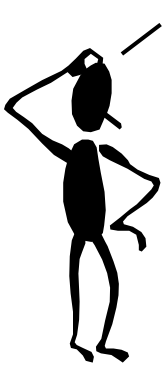
Recursion vs. Iteration

Recursion is often the simplest approach.

However, recursion can usually be replaced by iteration plus some storage for intermediate results.

```
unsigned int factorial(unsigned int n) {
    unsigned int ans = 1;
    for (int j = n; j > 1; j--) ans = ans * j;
    return ans;
}
```

Up Next



Thinking Recursively

DATA STRUCTURE