CSCI 262
Data Structures
8 – Recursion

RECURSION BASICS

Recursion
Recursion is defining something in terms of itself.
- 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
- 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: \( \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \)
  - 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.:
  ```cpp
  unsigned int factorial(unsigned int n) {
      if (n == 0) return 1;
      return n * factorial(n - 1);
  }
  ```

The Base Case
Note the first line of the `factorial` function:
```cpp
unsigned int factorial(unsigned int n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}
```
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

```c
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int x = factorial(5);
}
```

Factorial First Call

Push `factorial(5)`

```c
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int x = factorial(5);
}
```

Factorial Second Call

Push `factorial(4)`

```c
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int x = factorial(5);
}
```

Factorial Sixth Call

Push `factorial(0)` – Base case

```c
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int x = factorial(5);
}
```
unsigned factorial(unsigned n) {
    if (n == 0) return 1;
    return n * factorial(n-1);
}

int main() {
    int x = factorial(5);
}

Factorial Unwinding:
Pop factorial(0)

Factorial Unwinding:
Pop factorial(1)

Factorial Unwinding:
Pop factorial(2)

Factorial Unwinding:
Pop factorial(3)

Factorial Unwinding:
Pop factorial(4)

Factorial Unwinding:
Pop factorial(5)
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

Here's a recursive test function:

```cpp
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

```cpp
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:

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

- Recursion step doesn't reduce problem:

```cpp
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.

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

Up Next

- Wednesday, February 6
  - More recursion: thinking recursively, backtracking, minimax
- Friday, February 8
  - Lab 5 - TBD
  - Project 2 due
  - APT 3 assigned
- Week of February 11: Interview Grading