Lecture 16

CIS 208

Wednesday, March 16th, 2005

Extra Credit Due

Heap

- Dynamically allocated memory is in the heap. This is ‘global’ memory.
- When heap is full, malloc/calloc fail
- Heap is cleaned up by OS after program terminates (but you better still free it up)

Stack

- holds local variables and function parameters.
- Each function has it's own 'Activation Record' on the stack.
- Stack is of limited size (stack overflow)

functions calls on the stacks

1) function call is made.
2) input parameters are pushed onto stack
3) new activation record pushed on stack. (room for local variables)
4) new function knows location of parameters.

function is done

- activation record popped, param's popped.
- return value is pushed.

recursion

- Solving big problems by solving smaller problems.
- Method calls itself using smaller instance.

general template

```
return  func(input) {
   if input == base case;
        return answer;
  else return func(smaller input);
}
```

factorial

```
int fact(int in) {
if (in <= 1) return 1;
else return (in * fact(in-1));
}
```

Recursive methods

- Must provide stopping point, else program won't quit.
- Stop at small instance when recursion isn't necessary.

largest value in array

- 1st try: We know that the largest is either in the last element, or somewhere in the front.

```
int large(int AR[],int size) {
    if (size == 1) return AR[0];
   else return max(AR[size-1],large(AR,size-1));
}
```

don't need any extra variables. no local bookkeeping

largest value in array

- 2nd try: We know that largest value is either in first half, or 2nd half.

```
int large2(int in[]){
if (size == 1) return in[0];
if (size ==2 ) return max(AR[1],AR[2]);
else return max(large2(first half of AR),
                large2(second half of AR));
}
```

recursion vs iteration

- recursion: prettier solutions, easier to understand, easier to prove. Must use stack. Limited stack space
- iteration: no stack usage, only local variables. Generally faster. Harder to design/understand.

fibonacci sequence

- 1 1 2 3 5 8 13 21 etc…
- write a function for finding the nth fibonacci number.
- write both iterative and recursive.

last of recursion

- Use what works. Some solutions are better with iterative methods, others are better with recursive.

- linked data structures like recursion, but iterative can also work (linked list).

Just remember

Recursion reduces large problems into small problems.