Recursion

10-23-2002

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment? Don't let problems hang you up too long.
- What happens in C when a function calls another function? What about when a function calls itself? Can we utilize this to do interesting things?

Min Element of 2D Array

- Return does not mean print.

```
int minElem(int a[][10],int len) {
     int min=a[0][0],j,k;
     for(j=0; j<len; j++) {
          for(k=0; k<10; k++) {
               if(a[j][k]<min) {
                    min=a[j][k];
               }
          }
     }
     return min;
}
```

Recursive Functions

- We have looked at functions calling other functions, but what about when they call themselves? A function that calls itself is called a recursive function. They are heavily used in the mathematical theory of CS, but they can also be incredibly powerful tools in our code.
- There are some problems that are easy to solve with recursion, but extremely hard to solve without it.

Recursion and the Stack

- The real key to recursion is that one the stack, every call to the function gets its own stack frame and hence its own copy of all the local variables and arguments.
- This gives recursion some "memory" that it can go back to when functions return.
- Languages that don't use a stack for function calls can't be used for writing recursion.

Looping with Recursion

- The simplest use of recursion is simply to implement a loop. When doing this, typically a function takes a value as an argument and calls itself with a larger or smaller value of that argument.
- The recursive call has to be conditional. Otherwise it is like an inifinite loop, but this type causes a stack overflow and a segmentation fault.

Recursion in Multiple Directions

- Recursion is more powerful (and more useful) when the function contains two or more possible calls of itself. In this case, a loop can't be easily used as a substitute without major logic modifications.
- Drawing out the call path of these types of functions produces what computer scientists call a tree.
- The Fibonacci numbers are a simple example.

Recursive Fibonacci Numbers

- Notice how incredibly similar it is to the recurrence relationship that we started with. Unfortunately, in this case recursion is very inefficient because work is repeated frequently.

```
int f(int n) {
     if(n<3) return 1;
     return f(n-1)+f(n-2);
}
```

Flood Fill

- If you have used a painting package, you are familiar with the idea of "filling" the area around a specific location with a certain color.
- One way of doing this easily in code is with a recursive function that calls itself up to 4 times if adjacent pixels are blank.
- Let's write a bit of code for this.

Debuggers

- You can get help with finding runtime errors, and to some extent logic errors as well. Hopefully you have already used the practice of putting extra printf statements in code to help you figure out what it is doing. If you compile with the -g option you can also use a debugger.
- The debugger on these systems in gdb. You run "gdb a.out" (or whatever your program name is). Typing run at the prompt starts your program. You can pipe things in there too.

More on the Debugger

- When a fault occurs (could be you hitting ctrl-C) you can type in "where" to get a stack trace including line numbers.
- The command print will print out variable values.
- Use quit to get out.
- There are also many other powerful features in the debugger that allow you to set break points, return from functions, continue after breaks, etc. Use help to see these.

Minute Essay

- Is it clear how recursion works? Do you see how a loop can be converted to a recursive function that calls itself once? It is clear to you how recursive functions that call them selves more than once are different from loops?
- Assignment #5 is due today at midnight.