CS 137 Week 9
Fibonacci, More on Tail Recursion, Map and Filter

November 20th, 2017
Fibonacci Numbers

• An ubiquitous sequence named after Leonardo de Pisa (circa 1200) defined by

\[
\text{fib}(n) = \begin{cases} 
0 & \text{if } n == 0 \\
1 & \text{if } n == 1 \\
\text{fib}(n-1) + \text{fib}(n-2) & \text{otherwise}
\end{cases}
\]
Examples in Nature

- Plants, Pinecones, Sunflowers,
- Rabbits, Golden Spiral and Ratio connections
- Tool’s song Lateralu
- https://www.youtube.com/watch?v=wS7CZIJVxFY
#include <stdio.h>

int fib (int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-1)+fib(n-2);
}

int main () {
    printf("%d\n",fib(3));
    printf("%d\n",fib(10));
    // f_45 is largest that fits in integer.
    printf("%d\n",fib(45));
    return 0;
}
Fibonacci Call Tree

- The tree is really large, containing $O(2^n)$ many nodes (Actually grows with $\phi^n$ where $\phi$ is the golden ratio (1.618))
- Number of $\text{fib}(1)$ leaves is $\text{fib}(n)$
- Summing these is $O(\text{fib}(n))$
- Thus, the code on the previous slide runs in $O(\text{fib}(n))$ which is exponential!
Improvements

• This implementation of Fibonacci shouldn’t take this long - After all, by hand you could certainly compute more than $\text{fib}(45)$.

• We could change the code so that we’re no longer calling the stack each time, rather we’re using iterative structures.

• This would reduce the runtime to $O(n)$. 
int fib(int n){
    if(n==0) return 0;
    int prev = 0, cur = 1;
    for(int i=2; i<n; i++){
        int next = prev + cur;
        prev = cur;
        cur = next;
    }
    return cur;
}
Trace

<table>
<thead>
<tr>
<th>$n$</th>
<th>prev</th>
<th>cur</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Tail Recursive Fibonacci

```c
int fib_tr(int prev, int cur, int n){
    if(n==0) return cur;
    return fib_tr(cur, prev + cur, n - 1);
}
int fib(int n){
    if(n==0) return 0;
    return fib_tr(0, 1, n - 1);
}
```
Tail call elimination can reuse the activation record for each instance of fib_tr().
Counting Change

- Given an unlimited number of coins of specified denominations (say 5, 10, 25, 100, 200), count the number of unique ways to make change.
- For example, 5, 5, 10 and 5, 10, 5 are the same.
- Related to https://projecteuler.net/problem=31
Key Idea

- Take the number of ways to do this using all the coins up to coin \( i \) and then count all ways to do this without using coin \( i \) and decrease.
#include <stdio.h>

int count_change(int coin[], int n, int amount);

int main(void) {
    int coin[] = {5,10,25,100,200};
    const int n = sizeof(coin)/sizeof(coin[0]);
    printf("%d\n", count_change(coin,n,20));
    printf("%d\n", count_change(coin,n,200));
    return 0;
}

int count_change(
    int coin[], int n, int amount)
{
    if (amount == 0) return 1;
    if (amount < 0) return 0;
    if (n == 0) return 0;
    return count_change(coin, n, amount - coin[n - 1])
        + count_change(coin, n - 1, amount);
}
There are anonymous functions
Useful for simple functions needed only in one place
Unfortunately for us, C does not support them
However, C++11 (and further) does support this type of function (it’s a slightly different language but we will be able to get by)
Compile with

% g++ -std=c++11, name.cpp
Examples

```c
void (*f)(int) = 
  [](int i){printf("%d\n",i);};
```

- The first part says this is a pointer to a function that takes an integer and has no return value
- The `[]` denotes the start of a lambda function
- Within the braces denotes the function body.
- Return type of the body is inferred from the body (or explicitly using the `->` syntax)
- Calling `f(42);` will print out 42.
- Another way to call (type of `g` is inferred):
  ```c
  auto g = [](int i){printf("%d\n",i);};
  ```
Example with qsort

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int a[] = {2, -10, 14, 42, 11, -7, 0, 38};
    const int n = sizeof(a)/sizeof(a[0]);
    qsort(a, n, sizeof(int),
          [](const void *a, const void *b) {
              return *(int*)a - *(int*)b;
          }); // Change to flip order
    for (int i = 0; i < n; i++) printf("%d\n", a[i]);
    return 0;
}
```
Closure

Closure refers to a lambda function that captures variables from its containing scope. The following is only valid in

```c
#include <stdio.h>
auto return_fib(){
    int prev = 0, cur =1;
    return [prev, cur]() mutable{
        int next = prev + cur;
        prev = cur;
        cur =next;
        return prev;}
}
```

- `[prev, cur]` captures copies of the variables
- `mutable` makes the captured variables writeable.
```c
#include <stdio.h>

int main(void) {
    auto f1 = return_fib();
    auto f2 = return_fib();
    printf("%d %d %d\n", f1(), f1(), f1());
    printf("%d\n", f2());
    return 0;
}
```
Map and Reduce

- Map applies a transformation function to each element in an array, creating a new array.
- Reduce combines all elements into an array with one value.
Example

Below void *b is the destination array.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void map(const void *a, size_t n, size_t elem_a, void *b, size_t elem_b, void (*f)(const void *a, void *b)){
    for(int i=0; i<n; i++){
        f(a,b);
        a = (const char *)a + elem_a;
        b = (char *)b + elem_b;
    }
}
```
Example

Below, void *b is the reduction array.

```c
void reduce(const void *a, size_t n, size_t elem_a, void *b,
            void (*f)(const void *a, void *b)){
    for(int i=0; i<n; i++){
        f(a,b);
        a = (const char *)a + elem_a;
    }
}
```
Main

int main(void) {
    char *sentence[] = {"A", "day", "without", "sunshine", "is", "like", "night"};
    const int n = sizeof(sentence)/sizeof(sentence[0]);
    int lengths[n];
    map(sentence, n, sizeof(char*), lengths, sizeof(int), [](const void *a, void *b) {*(int *)b = strlen(*(char **)a);});
    int max = -1;
    reduce(lengths, n, sizeof(int), &max, [](const void *a, void *b) {
        if (*(int *)a > *(int *)b)
            *(int *)b = *(int *)a;
    });
    printf("%d\n", max);
    return 0;
}