Week 6 Lecture 1

Recursion

Recursion

- A function can use itself.
- Mathematical expression
  - n! = 1 * 2 * 3 … * n
  - = (n – 1)! * n
    - By associativity
- Same fact in C
  - fact(n) = n * fact(n – 1);

Using the fact in c

```c
int fact_rec(int n) {
  if (n <= 1) {
    return 1;
  } else {
    return fact_rec(n-1) * n;
  }
}
```

- factorial(n) == factorial(n-1) * n

Recursion v Iteration

```c
#include <stdio.h>

int fact_rec(int n) {
  if (n <= 1) {
    return 1;
  } else {
    return fact_rec(n-1) * n;
  }
}

int fact_iter(int n) {
  int acc = 1;
  for (int i = 1; i <= n; i++) {
    acc *= i;
  }
  return acc;
}

int main(int argv, char *argc[]) {
  int n = -1;

  printf("Enter integer> ");
  scanf("%d", &n);
  printf("Recursive factorial = %d\n",
         fact_rec(n));
  printf("Iterative factorial = %d\n",
         fact_iter(n));
  return 0;
}
fact.c (END)
```

```
> gcc -std=c11 -o fact fact.c
> ./fact
Enter integer> 5
Recursive factorial = 120
Iterative factorial = 120
> ./fact
Enter integer> 10
Recursive factorial = 3628800
Iterative factorial = 3628800
> ./fact
Enter integer> -10
Recursive factorial = 1
Iterative factorial = 1
> ./fact
Enter integer> 0
Recursive factorial = 1
Iterative factorial = 1
```

Why do we care

- It gives us a different way to reason about programs
  - What is the base case: fact(1) = 1
  - How do we reduce the size of the problem: fact(n) = fact(n-1) * n
- Here similar to iteration
  - Reducing from end instead of beginning.
- Efficient algorithms often result from reducing the size from the middle. i.e., Divide and reconquer.
  - This is much harder iteratively

Divide and Conquer

- Factorial requires a step for each number from one to n
  - It takes n steps
- If we can divide it in half, each step covers half the distance
  - Takes log n steps