ECE264: Advanced C Programming

Summer 2019

Week 3: Recursion

Recursion

- Function calling itself!

```
int factorial(int n)
{
      if(n == 0)
            return 1;
      else
            return n * factorial(n-1);
}
```

Recursion

- Better to think of recursion as a problem solving technique rather than a programming principle.
- A common pattern in problem solving:
  1. Break the problem into smaller problems
  2. Apply the same function to solve the smaller problems
  3. Use the solutions created in previous step to solve original problem

Recursion

- Is the pattern never ending?

  No.

- Repeating the process creates smaller and smaller problems. Eventually, the problem becomes *trivial* to solve.

  trivial problem = base case

Example - Factorial

- `n!` is just `n * (n-1)!`

Break the problem into smaller version of the same problem (step 1)

```
int factorial(int n)
{
      if(n == 0)
            return 1;
      else
            return n * factorial(n-1);
}
```

Example - Factorial

- call `factorial` again to solve the smaller problem

Solve the smaller problem by calling the same function (step 2)

```
int factorial(int n)
{
      if(n == 0)
            return 1;
      else
            return n * factorial(n-1);
}
```

Example - Factorial

- Multiply the result of previous step (calling `factorial(n-1)`) by n to find `factorial(n)`

Use the solution of the smaller problem to solve the original problem (step 3)

```
int factorial(int n)
{
      if(n == 0)
            return 1;
      else
            return n * factorial(n-1);
}
```

Example - Factorial

- The *base case* is simple: we know that `factorial(0) = 1`

```
int factorial(int n)
{
	if(n == 0)
		return 1;
	else
		return n * factorial(n-1);
}
```

Why recursive codes?

- Intuitive
  - Easier way to think of a solution
- Sometimes, the only way to effectively solve a problem!

Why recursive codes work?

- Think *inductively:*
  - Assume that the recursive function already works, but.. only on smaller problems than the original problem
  - Write recursive function for the original problem assuming it works
  - Write correct base case

Why recursive codes work?

- `Factorial` example*:*
  - Assume that `factorial(n-1)` works
  - If we have `(n-1)!` computing `n!` is easy:
    just multiply by n
  - Make sure that there exists a working base case: provide answer to the smallest argument passed to `factorial`

Divide-and-Conquer – A common recursive pattern

- Computing sum of array elements – toy example

```
int sum(int * arr, int nels)
{
      if (nels == 1)
            return arr[0];
      int sum1 = sum(arr, nels/2);
      int sum2 = sum(&arr[nels/2], (nels + 1)/2);
      return sum1 + sum2;
}
```

Divide-and-Conquer – A common recursive pattern

- Computing sum of array elements – toy example

```
int sum(int * arr, int nels)
{
      if (nels == 1)
            return arr[0];
      int sum1 = sum(arr, nels/2);
      int sum2 = sum(&arr[nels/2], (nels + 1)/2);
      return sum1 + sum2;
}
```

Divide-and-Conquer

- A problem can be broken into two or more smaller problems of similar or related type
- More realistic examples:

  Quicksort, Merge sort, finding closest pair of points

Recursion – observations

- Can have multiple base cases
  - Fibonacci series
- Tail recursion
  - `Factorial`

Using gdb to understand recursion

- Demo

```
#include<stdio.h>
int foo(int n)
{
        int retval = n;
        if (n == 0)
                return 1;
        retval = retval * foo(n-1);
        return retval;
}

int main()
{
        int x = foo(5);
        printf("foo(5)=%d\n",x);
}
```

Recursive vs. iterative codes

```
int sum(int * arr, int nels)
{
      if (nels == 1)
            return arr[0];
      int sum1 = sum(arr, nels/2);
      int sum2 = sum(&arr[nels/2], (nels + 1)/2);
      return sum1 + sum2;
}
```

```
int sum(int * arr, int nels)
{
      int total=0;
      for(int i=0;i<nels;i++)
            total += arr[i];
      return total;
}
```

Using gdb to understand recursion

- Tail recursion and its implementation advantages

Recursion - Exercise

- What happens in memory when recursion never terminates?