

# ECE 264 Advanced C Programming

2009/03/13

## Contents

1	Recursion (Continue)	1
2	Convert for to Recursion	2
3	Greatest Common Divisor (GCD)	3
4	2-Dimensional Recursion	5

## 1 Recursion (Continue)

Many problems are formulated as recursions and can be implemented as recursions. The Fibonacci numbers are an example:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n > 1. \end{cases} \quad (1)$$

n	0	1	2	3	4	5	6	7	8	9	10	11	12
Fibonacci(n)	0	1	1	2	3	5	8	13	21	34	55	89	144

It should not surprise you if we implement Fibonacci in this way:

```
#include <stdlib.h>
#include <stdio.h>
unsigned int Fibonacci(unsigned int n)
{
    if (n == 0) { return 0; }
    if (n == 1) { return 1; }
```

```

    return (Fibonacci(n - 1) + Fibonacci(n - 2));
}

int main(int argc, char * argv[])
{
    unsigned int n;
    if (argc < 2)
    {
        printf("need a number\n");
        return -1;
    }
    n = strtoul(argv[1], (char **)NULL, 10);
    printf("f(%d) = %d\n", n, Fibonacci(n));
    return 0;
}

```

I will talk more about this recursion later.

## 2 Convert for to Recursion

How do you compute the sum of  $0 + 1 + 2 + \dots + n$ ?

$$\sum_{i=0}^n i. \tag{2}$$

We can use for to compute the sum. We can also use recursion to compute the sum. In fact, we can compute it upward or downward, as in the case of factorial and Fibonacci.

```

#include <stdlib.h>
#include <stdio.h>
unsigned int computeSum1(unsigned int n)
{
    unsigned int cnt;
    unsigned int sum = 0;
    for (cnt = 0; cnt <= n; cnt++)
    {
        sum += cnt;
    }
    return sum;
}

```

```

unsigned int computeSum2(unsigned int n)
{
    if (n == 0)
        { return 0; }
    return (n + computeSum2(n - 1));
}

unsigned int computeSum3(unsigned int i, unsigned int n)
{
    if (i == n)
        { return n; }
    return (i + computeSum3(i + 1, n));
}

int main(int argc, char * argv[])
{
    printf("%d, %d, %d\n",
           computeSum1(100),
           computeSum2(100),
           computeSum3(0, 100));
    return 0;
}

```

The outputs from the three functions are the same, 5050.

### 3 Greatest Common Divisor (GCD)

Suppose  $a$  and  $b$  are two non-zero integers. An integer  $c$  is the *greatest common divisor* (GCD) of  $a$  and  $b$  if (1)  $c$  is a common divisor of  $a$  and  $b$  and (2)  $c$  is a multiple of any common divisor of  $a$  and  $b$ . For example,  $\text{GCD}(6, 9) = 3$ ,  $\text{GCD}(10, 25) = 5$ ,  $\text{GCD}(5, 19) = 1$ ,  $\text{GCD}(21, 84) = 21$ .

One algorithm to find the GCD of two integer is

```

if ((a % b) == 0)
{
    GCD(a, b) = b;
}
else

```

```

{
    GCD(a, b) = GCD(b, a % b);
}

```

How do we implement it using recursion?

```

#include <stdlib.h>
#include <stdio.h>
int gcd(int a, int b)
{
    printf("(a, b) = (%d, %d)\n", a, b);
    if ((a % b) == 0)
        { return b; }
    /* else not necessary since the previous if uses return */
    return gcd(b, a % b);
}

int main(int argc, char * argv[])
{
    int val1;
    int val2;
    if (argc < 3)
        {
            printf("need two integers\n");
            return -1;
        }
    val1 = strtol(argv[1], (char **)NULL, 10);
    val2 = strtol(argv[2], (char **)NULL, 10);
    printf("GCD(%d,%d) = %d\n", val1, val2, gcd(val1, val2));
    return 0;
}

/*
    (a, b) = (5, 15)
    (a, b) = (15, 5)
    GCD(5,15) = 5

    (a, b) = (15, 25)
    (a, b) = (25, 15)
    (a, b) = (15, 10)
    (a, b) = (10, 5)
    GCD(15,25) = 5

```

```

(a, b) = (35, 25)
(a, b) = (25, 10)
(a, b) = (10, 5)
GCD(35,25) = 5

```

```

(a, b) = (42, 35)
(a, b) = (35, 7)
GCD(42,35) = 7

```

```

(a, b) = (42, 120)
(a, b) = (120, 42)
(a, b) = (42, 36)
(a, b) = (36, 6)
GCD(42,120) = 6

```

\* /

## 4 2-Dimensional Recursion

Consider this function  $f(n, k)$  defined for two non-zero integers  $n$  and  $k$ :

$$f(n, k) = \begin{cases} 0 & \text{if } n = 0 \text{ or } k = 0, \\ 1 & \text{if } n = k \neq 0, \\ n \cdot f(n-1, k) & \text{if } n > k, \\ \frac{f(n, k-1)}{k} & \text{otherwise,} \end{cases} \quad (3)$$

here we use **integer division**. If  $a$  and  $b$  are two integer and  $a < b$ ,  $\frac{a}{b}$  is zero.

```

#include <stdlib.h>
#include <stdio.h>
int compute(int a, int b)
{
    printf("(a, b) = (%d, %d)\n", a, b);
    if ((a == 0) || (b == 0)) { return 0; }
    if (a == b) { return 1; }
    if (a > b) { return a * compute(a - 1, b); }
    return compute(a, b - 1) / b;
}

int main(int argc, char * argv[])

```

```

{
    int val1;
    int val2;
    if (argc < 3)
    {
        printf("need two integers\n");
        return -1;
    }
    val1 = strtol(argv[1], (char **)NULL, 10);
    val2 = strtol(argv[2], (char **)NULL, 10);
    printf("compute(%d,%d) = %d\n", val1, val2, compute(val1, val2));
    return 0;
}

/*
(a, b) = (6, 4)
(a, b) = (5, 4)
(a, b) = (4, 4)
compute(6,4) = 30

(a, b) = (8, 4)
(a, b) = (7, 4)
(a, b) = (6, 4)
(a, b) = (5, 4)
(a, b) = (4, 4)
compute(8,4) = 1680

(a, b) = (4, 8)
(a, b) = (4, 7)
(a, b) = (4, 6)
(a, b) = (4, 5)
(a, b) = (4, 4)
compute(4,8) = 0

(a, b) = (4, 2)
(a, b) = (3, 2)
(a, b) = (2, 2)
compute(4,2) = 12

*/

```