

Introduction to Computer and Lab

Homework #7

Due date: Mar 26, 2016

학번: 201404051

이름: 정 용 석

1. 수의 합 구하기

1.1 Solution

간단한 문제로 딱히 설명할 것이 없다. 입력 정수 N 이 주어졌을 때, 재귀를 통하여 현재 수 N 과 1 이 줄어드는 $N-1$ 의 합을 재귀 함수를 통해 구현하였다. N 에서부터 1 까지 도착했을 때, 재귀 함수는 1 을 리턴함으로서 현재까지 호출된 모든 함수 값을 리턴함으로서 수의 합을 출력한다.

1.2. Source code

```
long long sum_recursion(long long n) //수의 합 재귀 함수
{
    //재귀 함수 종료 조건 n == 1
    if (n == 1)
        return n;
    return n + sum_recursion(n - 1);
}
```

1.3. Result (snapshot)

```
42
903

2
3

123
7626

1000
500500

2000
2001000

654
214185
```

2 홀수의 합 출력

2.1. Solution

문제 1 과 거의 동일하지만 조건이 조금 다르다. 입력 받은 정수 N 이 일단 홀수 혹은 짝수일 때 리턴 해야하는 재귀 함수 문이 달라진다. N 이 홀수일 때는 간단하게 현재 수 n 과 그 다음 홀수인 $n-2$ 를 차례로 더해주면 된다. 하지만 N 이 짝수이면 현재 수 n 에서 그 다음 홀수인 $n-1$ 과 그 다음 홀수인 $n-3$ 을 더하는 식의 재귀 함수를 만들어야 한다. 또한 홀수일 때와 짝수일 때 모두 종료 조건 문이다 다르다. 홀수일 경우 n 은 1 을 마지막으로 갖게 되므로 여기서 끝을 맺고, n 이 짝수일 경우에는 0 이 되기에 그에 맞는 종료 문을 만들어 주면 된다.

2.2. Source code

```
long long oddSum_recursion(long long n)
{
    //n == 1일 때(즉, 시작 n이 홀 수일 때 가능) 재귀함수 종료
    if (n == 1)
        return n;
    //n < 1일 때(n이 짝수일 때) 재귀 함수 종료
    else if (n < 1)
        return 0;
    //시작 n이 짝수일 때 1을 빼서 홀수로 만들기
    if (n % 2 == 0)
        return (n - 1) + oddSum_recursion(n - 3);
    return n + oddSum_recursion(n - 2);
}
```

2.3. Result (snapshot)

```
725
131769

12
36

5
9

6
9

6464
10445824
```

3 2^N 계산하기

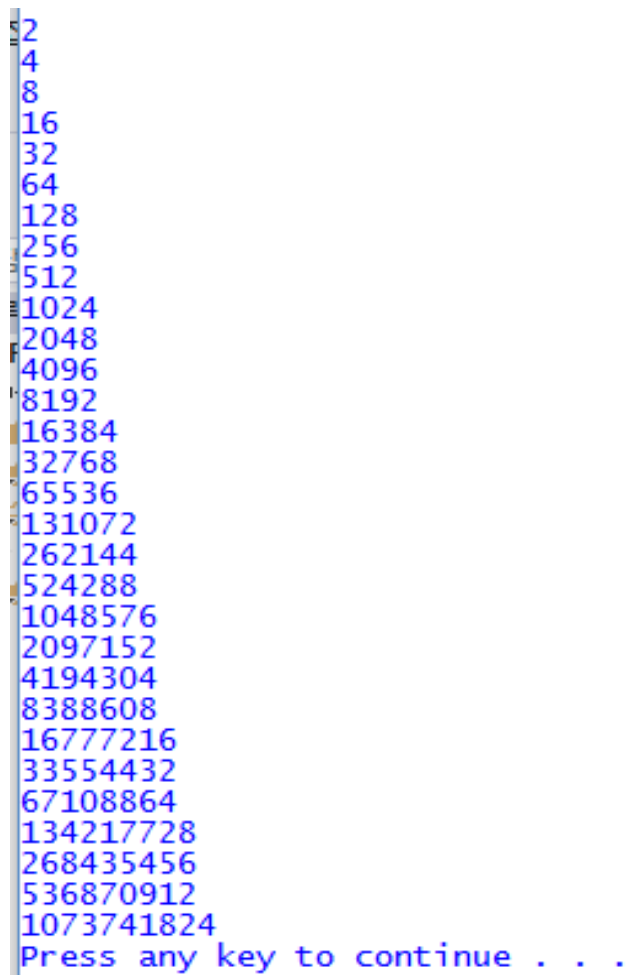
3.1. Solution

2^N 이라 하면 2^0 부터 2 를 N 번 만큼 곱하면 된다는 소리이고, 그 말인 즉 그만큼 재귀 함수를 호출해서 곱하면 된다는 말이다. 따라서 $n-1$ 에 대한 재귀 함수에 2 를 지속적으로 곱하고, 탈출 조건으로 n 이 0 일 때, $2^0 = 1$ 이므로 1 을 리턴함으로서 호출 종료를 시키면 된다.

3.2. Source code

```
long long exp_recursion(long long n)
{
    //2^0일 경우 재귀 종료, 2^0 == 1
    if (n == 0)
        return 1;
    return 2 * exp_recursion(n - 1);
}
```

3.3. Result (snapshot)



A screenshot of a program's output, showing a vertical list of powers of 2. The numbers are displayed in a blue monospace font. The list starts with 2 and continues up to 1073741824. At the bottom, there is a prompt 'Press any key to continue . . .'. The numbers are: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824.

Power of 2
2^0
2^1
2^2
2^3
2^4
2^5
2^6
2^7
2^8
2^9
2^{10}
2^{11}
2^{12}
2^{13}
2^{14}
2^{15}
2^{16}
2^{17}
2^{18}
2^{19}
2^{20}

4. Fibonacci 수열 출력

4.1. Solution

피보나치의 수열 같은 경우는 수열의 순서가 첫 번째와 두 번째가 모두 1 그리고 나머지는 전 두수의 합으로 이루어져 있다. 따라서 1 번째와 2 번째 수가 주어지면 재귀 함수를 통해 다음 수를 구하게 하면 된다.

4.2. Source code

```
long long fib_recursion(long long n)
{
    //피보나치 수열의 첫번째와 2번째 수의 값
    if (n == 1 || n == 2)
        return 1;
    return fib_recursion(n - 1) + fib_recursion(n - 2);
}
```

4.3. Result (snapshot)

```
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040

Press any key to continue . . .
```

5 숫자를 거꾸로 출력

5.1. Solution

여러가지 방법이 있겠지만 필자는 다음과 같은 방법으로 구현하였다. 일단 각 자리 수를 구하는 방법은 항상 해왔듯이 %연산자를 이용하였다. 이 연산자를 이용하면 일의 자리의 수부터 각각 구할 수 있다. 숫자를 거꾸로 출력한다는 것은 일의 자리의 수부터 구한 수를 반대쪽 자리의 수로 만들어야 한다는 점이다. 따라서 일의 자리의 수를 구함과 동시에 현재 수의 가장 큰 자리의 수의 위치를 구했다. N 이 10 으로 몇 번 나누어지는지에 따라 가장 첫 자리가 10^{base} 자리임을 알 수 있고, 이를 맨 처음에 구하는 일의 자리에 곱하여 일의 자리를 가장 첫 자리의 수로 탈바꿈 시켜주는 작업을 했다. 그리고 재귀 함수의 진행 문으로 다음 자리를 구할 수 있도록 $n/10$ 으로 하고 종료 문으로 n 이 0 일 때, 0 을 리턴함으로서 종료 시킨다.

5.2. Source code

```
int reverse_Number(int n) {
    int add;
    int temp = n;
    int base = 1;

    //각 자리별 자리 수 구하기
    while (temp > 10) {
        temp = temp / 10;
        base *= 10;
    }
    //종료문
    if (n == 0)
        return 0;

    //n이 일의 자리 수 일때는 그대로 더한다.
    else if (n < 10)
        add = n;
    //n의 현재 자리를 반대 쪽 자리의 수로 변환
    else
        add = (n % 10) * base;;
    return add + reverse_Number(n / 10);
}
```

5.3. Result (snapshot)

```
123456
654321

321654
456123

545454
454545

789852
258987

1234567
7654321

654321
123456
```

6. 숫자의 자리수의 합

6.1 Solution

5 번 문제와 거의 메커니즘이 비슷하지만 훨씬 간단하다. 각 자리 수를 구하고 구한 수를 차례로 더해가면 된다. 딱히 설명이 불필요하다고 생각한다.

6.2 Source code

```
int digitSum_recursion(int n){
    int temp = n;
    int add;
    //각 자리수 구하기
    add = temp % 10;
    //종료문
    if (n == 0)
        return 0;
    return add + digitSum_recursion(n / 10);
}
```

6.3 Result (snapshot)

```
12
3

123
6

1234
10

789852
39

6543
18
```

7. 이진수의 1 의 개수 출력

7.1 Solution

5 번 6 번과 마찬가지로 자리 수만 구해서 더해주면 된다. 조금 다른 점은 16 자리 이진수를 받기 때문에 long long 자료형으로 입출력을 진행해야 한다. 자리 수가 1 일 때만 1 씩 늘려주는 형식의 재귀 함수를 구현하면 된다.

7.2 Source code

```
int binaryNum_recursion( long long n) {  
    int add;  
    long long temp = n;  
    //종료문  
    if (n == 0)  
        return 0;  
    //이진수의 자리수가 0이면 0 리턴, 1이면 1 리턴  
    if (temp % 10 == 1)  
        add = 1;  
    else  
        add = 0;  
    return add + binaryNum_recursion(n / 10);  
}
```

7.3 Result (snapshot)

```
11010101  
5  
  
11111  
5  
  
11111111111111111  
17  
  
10101010  
4  
  
00000001  
1  
  
0000000000  
0
```

8. 계단 오르기

8.1 Solution

이 문제의 경우 재귀 함수를 구현하기 위한 패턴을 알아내야 한다. 보면 계단이 1 개일 때의 경우의 수는 (1)로 1 개이고, 2 개일 때는 (1,1), (2)로 2 개이다. 3 일 경우는 3 가지이고, 4 일 경우는 5 가지이다. 5 일 경우는 8 가지로 잘 보면 피보나치 수열과 비슷하다. 구해야 하는 수가 이전 2 개의

경우의 수의 합이 된다. 따라서 피보나치 수열과 동일하게 첫 2 개의 조건을 주고 재귀를 통해 이를 구현하면 간단하다.

8.2 Source code

```
long long stair_recursion(long long n)
{
    //계단 수가 1일 때 경우의 수 1, 2일 때 경우의 수 2
    // *3부터는 계단 수가 1일 때와 2일 때 경우의 수의 합이다.
    if (n == 1)
        return 1;
    else if (n == 2)
        return 2;
    else
        return stair_recursion(n - 1) + stair_recursion(n - 2);
}
```

8.3 Result (snapshot)

```
12
233

30
1346269

13
377

18
4181

31
2178309
```