

<피보나치 수열 재귀,반복 함수 비교>

20221057 권관우

코드

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

typedef unsigned long ULONG;

ULONG Fibonacci_Repetition_count = 0;
ULONG Fibonacci_Recursion_count = 0;

//반복문으로 표현한 피보나치 수열
ULONG Fibonacci_Repetition(int N)
{
    int i;
    ULONG Result;
    ULONG* FibonacciTable;

    if (N == 0 || N == 1)
        return (ULONG)N;

    FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (N + 1));
    FibonacciTable[0] = 0;
    FibonacciTable[1] = 1;

    for (i = 2; i <= N; i++) {
        FibonacciTable[i] = FibonacciTable[i - 1] + FibonacciTable[i - 2];
        Fibonacci_Repetition_count++;
    }

    Result = FibonacciTable[N];
    free(FibonacciTable);
    return Result;
}

//재귀함수로 표현한 피보나치 수열
ULONG Fibonacci_Recursion(int N)
{
    Fibonacci_Recursion_count++;
    ULONG Result;
    ULONG* FibonacciTable;
    if (N == 0 || N == 1)
        return (ULONG)N;

    FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (N + 1));
    FibonacciTable[0] = 0;
    FibonacciTable[1] = 1;

    if (N >= 2)
        FibonacciTable[N] = Fibonacci_Recursion(N - 1) + Fibonacci_Recursion(N - 2);
```

```

    Result = FibonacciTable[N];
    free(FibonacciTable);
    return Result;
}

int main(void)
{
    int N = 3;
    ULONG Result;
    clock_t start, stop;
    double duration;

    while (N <= 40) {
        start = clock();
        Fibonacci_Repetition_count = 0;
        Result = Fibonacci_Repetition(N);
        stop = clock();
        duration = ((double)(stop - start)) / CLOCKS_PER_SEC;
        printf("Fibonacci_Repetition(%d)=%lu(time:%lf),step:%d\n",N, Result, duration, Fibonacci_Repetition_count);
        printf("\n");

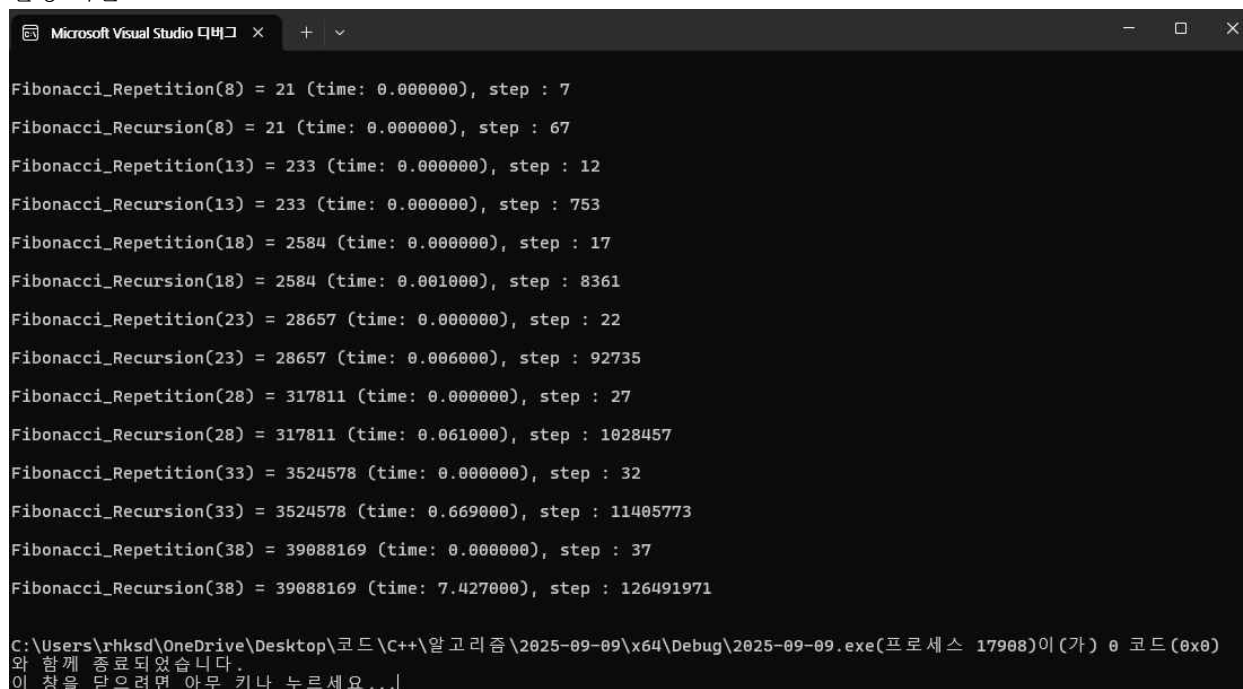
        start = clock();
        Fibonacci_Recursion_count = 0;
        Result = Fibonacci_Recursion(N);
        stop = clock();
        duration = ((double)(stop - start)) / CLOCKS_PER_SEC;
        printf("Fibonacci_Recursion(%d)=%lu(time:%lf),step:%d\n",N, Result, duration, Fibonacci_Recursion_count);
        printf("\n");

        N = N + 5;
    }

    return 0;
}

```

실행 화면



```

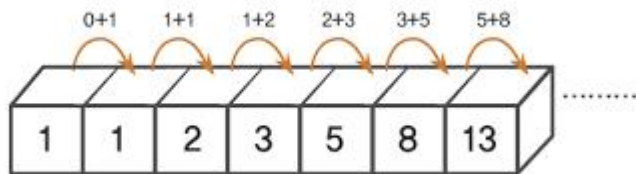
Microsoft Visual Studio 디버그
Fibonacci_Repetition(8) = 21 (time: 0.000000), step : 7
Fibonacci_Recursion(8) = 21 (time: 0.000000), step : 67
Fibonacci_Repetition(13) = 233 (time: 0.000000), step : 12
Fibonacci_Recursion(13) = 233 (time: 0.000000), step : 753
Fibonacci_Repetition(18) = 2584 (time: 0.000000), step : 17
Fibonacci_Recursion(18) = 2584 (time: 0.001000), step : 8361
Fibonacci_Repetition(23) = 28657 (time: 0.000000), step : 22
Fibonacci_Recursion(23) = 28657 (time: 0.006000), step : 92735
Fibonacci_Repetition(28) = 317811 (time: 0.000000), step : 27
Fibonacci_Recursion(28) = 317811 (time: 0.061000), step : 1028457
Fibonacci_Repetition(33) = 3524578 (time: 0.000000), step : 32
Fibonacci_Recursion(33) = 3524578 (time: 0.669000), step : 11405773
Fibonacci_Repetition(38) = 39088169 (time: 0.000000), step : 37
Fibonacci_Recursion(38) = 39088169 (time: 7.427000), step : 126491971

C:\Users\rhksd\OneDrive\Desktop\코드\C++\알고리즘\2025-09-09\x64\Debug\2025-09-09.exe(프로세스 17908)이 (가) 0 코드(0x0)
와 함께 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요...|

```

반복문으로 표현한 피보나치 코드와 비교하면 재귀로 표현한 코드는 시간차이가 확실하다.
 이러한 차이는 재귀함수 코드의 구조로부터 나타나는데
 지금부터 반복문의 코드 진행 방식과 재귀 함수의 진행방식을 표현하고 재귀함수의 단점을 개선한 코드를 작성하겠다.

반복문의 피보나치 수열 구하는 진행 방식은



위 사진과 유사하다.

우선 함수시작시 몇 번째 피보나치 수를 구할 것인지 받아서

0과 1이라면 그 값이기에 그대로 반환을 해주고

아니라면(이상이라면)

```
FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (N + 1));
```

위 코드로 피보나치 수를 저장할 배열을 만들고

기본적으로 배열의 0번에 0과 1번에 1을 저장한다. 이후 저장된 값을 불러와서 다음 공간의 값을 채운다.

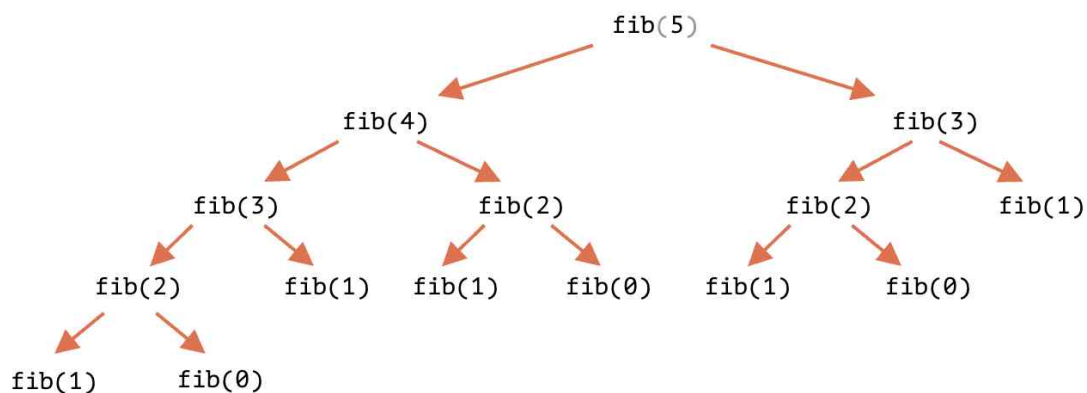
진행 방식을 생각해 봤을 때 딱히 낭비되는 연산은 없다. 반복횟수도 납득 가능하다.

반면에 재귀함수를 보자 벌써부터 비효율적인게 눈에 띈다.

우선 재귀함수가 피보나치 수를 구하는 방식을 들여다볼 필요가 있다.

```
if (N >= 2)
```

```
FibonacciTable[N] = Fibonacci_Recursion(N - 1) + Fibonacci_Recursion(N - 2);
```



해당 방식으로 값을 리

턴하여 더하는 방식이다.

따라서 재귀함수의 연산에서 아래의 코드처럼 배열을 생성하여 저장하고 하는 방식은 쓸모가 없다.

```
FibonacciTable = (ULONG*)malloc(sizeof(ULONG) * (N + 1));
```

```
FibonacciTable[0] = 0;
```

```
FibonacciTable[1] = 1;
```

해당 코드 없이도 잘 돌아간다. 왜 있는걸까,

없이도 피보나치 재귀함수에 들어간 값이 1이나 0이되면 반환되어 더해지는 형식으로 결국 처음 시작한 함수에서 저장될 공간 하나만 있으면 작동한다.

재귀 함수를

```
ULONG Fibonacci_Recursion(int N)
```

```
{
```

```
Fibonacci_Recursion_count++;
```

```
ULONG Result;
```

```
ULONG* FibonacciTable;
```

```
if (N == 0 || N == 1)
```

```
return (ULONG)N;
```

```

    if (N >= 2)
        Result = Fibonacci_Recursion(N - 1) + Fibonacci_Recursion(N - 2);

    return Result;
}

```

이렇게만 바꾸어도 실행 시간이

```

Fibonacci_Recursion(8) = 21 (time: 0.000000), step : 67
Fibonacci_Repetition(13) = 233 (time: 0.000000), step : 12
Fibonacci_Recursion(13) = 233 (time: 0.000000), step : 753
Fibonacci_Repetition(18) = 2584 (time: 0.000000), step : 17
Fibonacci_Recursion(18) = 2584 (time: 0.000000), step : 8361
Fibonacci_Repetition(23) = 28657 (time: 0.000000), step : 22
Fibonacci_Recursion(23) = 28657 (time: 0.001000), step : 92735
Fibonacci_Repetition(28) = 317811 (time: 0.000000), step : 27
Fibonacci_Recursion(28) = 317811 (time: 0.011000), step : 1028457
Fibonacci_Repetition(33) = 3524578 (time: 0.000000), step : 32
Fibonacci_Recursion(33) = 3524578 (time: 0.101000), step : 11405773
Fibonacci_Repetition(38) = 39088169 (time: 0.000000), step : 37
Fibonacci_Recursion(38) = 39088169 (time: 1.135000), step : 126491971

C:\Users\rhksd\OneDrive\Desktop\코드\C++\알고리즘\2025-09-09\x64\Debug\2025-09-09.exe(프로세스 19512)이(가) 0 코드(0x0)
와 함께 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요...

```

1.135로 약 6이나 절약된다.

그러나 이렇게 하더라도 불필요한 연산이 진행되기는 마찬가지다.

구한 값을 다시 구하는 연산이 들어가기 때문에 이러한 불필요한 연산을 없애기위해.

이미 구한 값이 있다면 그 상태로 바로 반환되게 하여 연산 수를 줄일 생각이다.

그렇려면 저장을 하기 위해 맨 초기에 있던 재귀함수의 배열방식이 필요하다.

하지만 함수내에서 동적으로 배열을 생성하는 행위는 낭비이기에 외부에서 N만큼의 크기에 해당하는 배열을 하나 만들고 함수에 진입하여 저장하는 방식으로 하겠다.

구현한 코드가 아래 코드다

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef unsigned long ULONG;
ULONG* FibonacciTable;
ULONG Fibonacci_Recursion_count = 0;

```

//재귀함수를 통한 피보나치 수열

```

ULONG Fibonacci_Recursion(int N)
{
    ULONG Result;
    if (N == 0 || N == 1)
        return (ULONG)N;
    else if (FibonacciTable[N] != 0) {
        return FibonacciTable[N];
    }
    else {

```

```

        FibonacciTable[N] = Fibonacci_Recursion(N - 1) + Fibonacci_Recursion(N - 2);
    }
    Fibonacci_Recursion_count++;
    Result = FibonacciTable[N];
    return Result;
}
int main(void)
{
    int N = 3;
    ULONG Result;
    clock_t start, stop;
    double duration;
    while (N <= 40) {
        start = clock();
        FibonacciTable = (ULONG*)calloc(N + 1, sizeof(ULONG));
        FibonacciTable[0] = 0;
        FibonacciTable[1] = 1;
        Fibonacci_Recursion_count = 0;
        Result = Fibonacci_Recursion(N);
        free(FibonacciTable);
        stop = clock();
        duration = ((double)(stop - start)) / CLOCKS_PER_SEC;
        printf("Fibonacci_Recursion(%d)=%lu(time:%lf)step%d\n", N, Result, duration, Fibonacci_Recursion_count);
        printf("\n");
        N = N + 5;
    }
    return 0;
}

```

함수 안에서 테이블을 만들어버리면 그 테이블은 계속해서 생성되고 하는 불필요한 연산이 너무 많아서 전역변수로 선언하여 메인함수와 재귀함수에서 호출이 가능하게끔 하고 재귀함수로 진입 전에 테이블의 크기를 선언하고 calloc을 통해 0으로 초기화 해주면서 기본 값을 세팅해준다. 또한 한번 구한 피보나치 값을 또 구하지 않기위해

```

else if (FibonacciTable[N] != 0)
    return FibonacciTable[N];

```

이 조건으로 검사를 하여 구했었다면 바로 반환하여 불필요한 연산을 최소화 한다. 아래 실행 사진

```

Microsoft Visual Studio 디버그
Fibonacci_Recursion(8) = 21 (time: 0.000000) step 7
Fibonacci_Repetition(13) = 233 (time: 0.000000) step 12
Fibonacci_Recursion(13) = 233 (time: 0.000000) step 12
Fibonacci_Repetition(18) = 2584 (time: 0.000000) step 17
Fibonacci_Recursion(18) = 2584 (time: 0.000000) step 17
Fibonacci_Repetition(23) = 28657 (time: 0.000000) step 22
Fibonacci_Recursion(23) = 28657 (time: 0.000000) step 22
Fibonacci_Repetition(28) = 317811 (time: 0.000000) step 27
Fibonacci_Recursion(28) = 317811 (time: 0.000000) step 27
Fibonacci_Repetition(33) = 3524578 (time: 0.000000) step 32
Fibonacci_Recursion(33) = 3524578 (time: 0.000000) step 32
Fibonacci_Repetition(38) = 39088169 (time: 0.000000) step 37
Fibonacci_Recursion(38) = 39088169 (time: 0.000000) step 37

C:\Users\rhksd\OneDrive\Desktop\코드\C++\알고리즘\2025-09-09\x64\Debug\2025-09-09.exe(프로세스 22820)이 (가) 0 코드(0x0)
와 함께 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요...

```