

# Data Structures (3200)

## Homework 1 Report

Yulwon Rhee (202211342)

Department of Computer Science and Engineering, Konkuk University

**Problem 1** 수업 시간에 배운, factorial 프로그램을 recursion version과 iteration version으로 각각 구현하고, 입력 값을 1-10까지 증가시키면서 화면에 출력되게 실행하시오. n값은 scanf으로 입력받지 말고, main 함수에서 parameter로 전달해서 수행하시오. (의도: 구현이 정확하게 되었는지를 보는 문제입니다.)

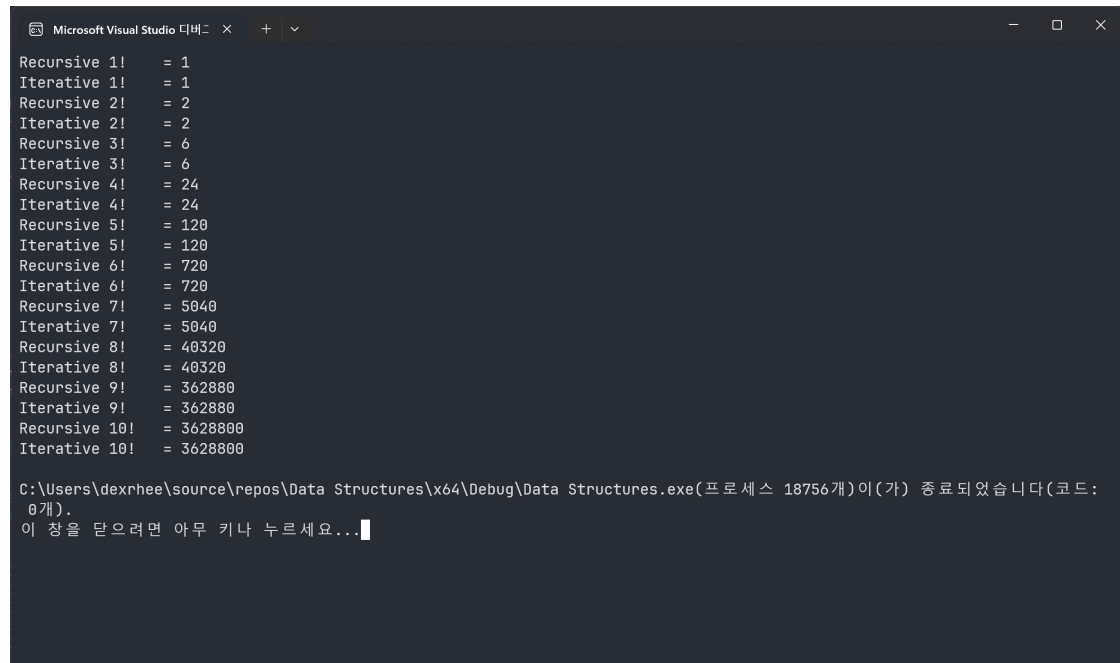
*Solution.* 다음은 recursive version과 iteration version으로 factorial function을 구현한 소스코드입니다. (제출한 파일의 8-13번 라인)

```
1 // 1-1. Recursion Version of the Factorial
2 long long factorialRecursive(int n) {
3     return n == 0 || n == 1 ?
4         1 :
5         n * factorialRecursive(n - 1);
6 }
7
8 // 1-2. Iterative Version of the Factorial
9 long long factorialIterative(int n) {
10     long long result = 1;
11     for (int i = 1; i <= n; i++) {
12         result *= i;
13     }
14     return result;
15 }
```

위 함수의 parameter n을 1부터 10까지 증가시키며 테스트하기 위해 사용한 코드는 다음과 같습니다. (제출한 파일의 44-48번 라인)

```
1 // 1-3. Factorial 1 to 10
2 for (int i = 1; i <= 10; i++) {
3     printf("Recursive %d!\t= %lld\n", i, factorialRecursive(i));
4     printf("Iterative %d!\t= %lld\n", i, factorialIterative(i));
5 }
```

위의 코드를 실행한 결과는 다음과 같습니다.



```
Microsoft Visual Studio 디버깅 x + v
Recursive 1! = 1
Iterative 1! = 1
Recursive 2! = 2
Iterative 2! = 2
Recursive 3! = 6
Iterative 3! = 6
Recursive 4! = 24
Iterative 4! = 24
Recursive 5! = 120
Iterative 5! = 120
Recursive 6! = 720
Iterative 6! = 720
Recursive 7! = 5040
Iterative 7! = 5040
Recursive 8! = 40320
Iterative 8! = 40320
Recursive 9! = 362880
Iterative 9! = 362880
Recursive 10! = 3628800
Iterative 10! = 3628800

C:\Users\dexrhee\source\repos\Data Structures\x64\Debug\Data Structures.exe(프로세스 18756개)이(가) 종료되었습니다(코드:
07h).
이 창을 닫으려면 아무 키나 누르세요...
```

**Problem 2** 피보나치 수열에 대한 프로그램도 recursion version과 iteration version을 각각 구현하고, 입력 값을 1-10까지 증가시키면서 화면에 출력되게 실행하시오. (의도: 구현이 정확하게 되었는지를 보는 문제입니다.)

*Solution.* 다음은 recursion version과 iteration version으로 fibonacci function을 구현한 소스코드입니다. (제출한 파일의 24-29번 라인)

```

1 // 2-1. Recursive Version of the Fibonacci
2 int fibonacciRecursive(int n) {
3     return n == 0 || n == 1 ?
4         n :
5         fibonacciRecursive(n - 2) + fibonacciRecursive(n - 1);
6 }
7
8 // 2-2. Iterative Version of the Fibonacci
9 int fibonacciIterative(int n) {
10     int a = 0, b = 1, c;
11     if (n == 0) return a;
12     for (int i = 2; i <= n; i++) {
13         c = a + b;
14         a = b;
15         b = c;
16     }
17     return b;
18 }

```

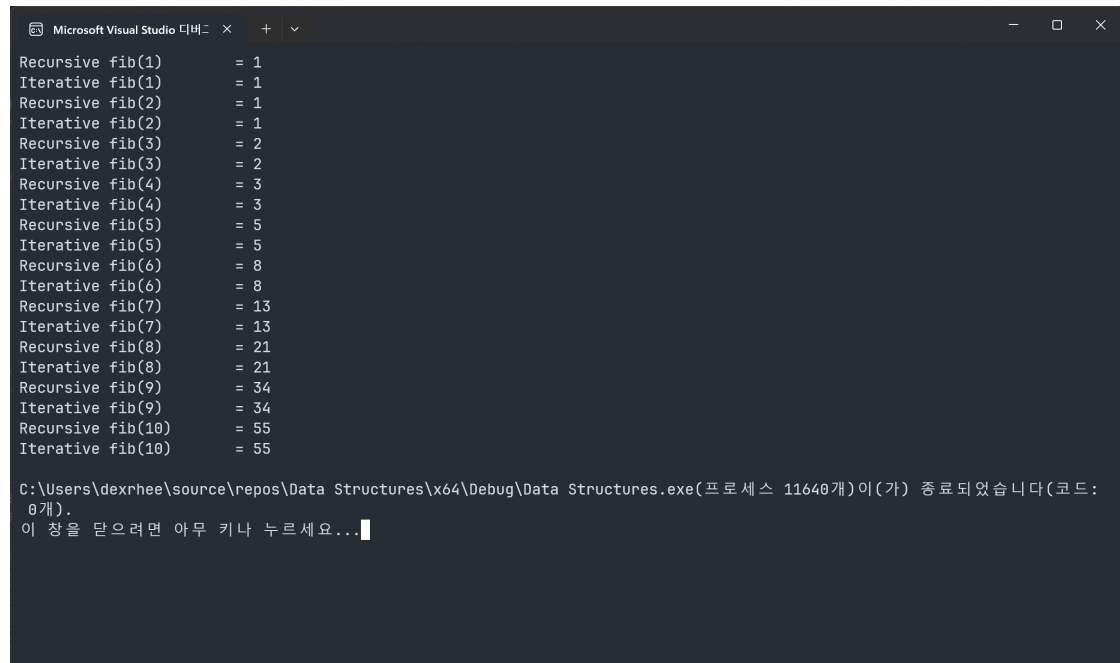
위 함수의 parameter n을 1부터 10까지 증가시키며 테스트하기 위해 사용한 코드는 다음과 같습니다. (제출한 파일의 51-55번 라인)

```

1 // 2-3. Fibonacci 1 to 10
2 for (int i = 1; i <= 10; i++) {
3     printf("Recursive fib(%d)\t= %d\n", i, fibonacciRecursive(i));
4     printf("Iterative fib(%d)\t= %d\n", i, fibonacciIterative(i));
5 }

```

위의 코드를 실행한 결과는 다음과 같습니다.



```
Microsoft Visual Studio 디버깅
Recursive fib(1)      = 1
Iterative fib(1)      = 1
Recursive fib(2)      = 1
Iterative fib(2)      = 1
Recursive fib(3)      = 2
Iterative fib(3)      = 2
Recursive fib(4)      = 3
Iterative fib(4)      = 3
Recursive fib(5)      = 5
Iterative fib(5)      = 5
Recursive fib(6)      = 8
Iterative fib(6)      = 8
Recursive fib(7)      = 13
Iterative fib(7)      = 13
Recursive fib(8)      = 21
Iterative fib(8)      = 21
Recursive fib(9)      = 34
Iterative fib(9)      = 34
Recursive fib(10)     = 55
Iterative fib(10)     = 55

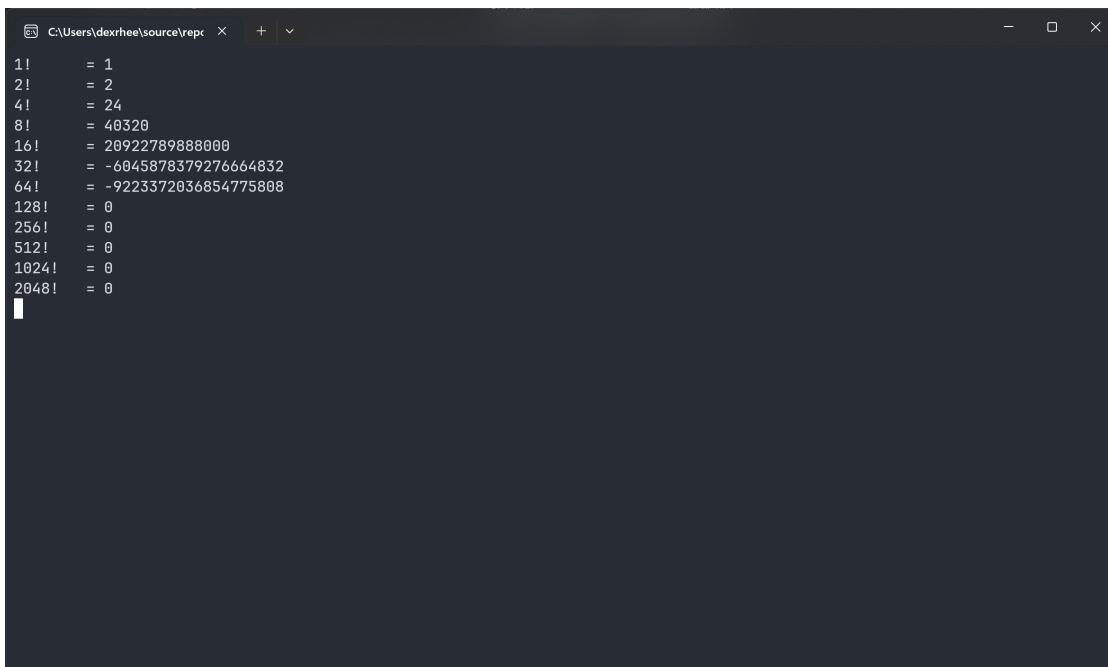
C:\Users\dexrhee\source\repos\Data Structures\x64\Debug\Data Structures.exe(프로세스 11640개)이(가) 종료되었습니다(코드:
07h).
이 창을 닫으려면 아무 키나 누르세요...
```

**Problem 3** Factorial recursion version의 경우,  $n$ 이 증가함에 따라 어느 정도의  $n$ 에서까지는 실행 (예, 1000, 2000, 3000, 4000. 수행 컴퓨터마다 다를 수 있음)이 잘 되지만, 어느 정도 이후(약 5000. OS따라 Compiler에 따라 한계값이 다를 수 있음)에는 “stack overflow”로 수행이 안 됨을 보이시오. 단, `factorial(1000)` 등은 결과값이 너무 커서, 그 결과값을 **unsigned long long** 타입으로 설정하여도 overflow됨으로 결과값은 틀려도 됩니다. (의도: stack overflow 로 인해 run time error로 수행이 멈춘다는 것을 확인하는 것입니다.)

*Solution.* 위 상황을 테스트하기 위해 사용한 코드는 아래와 같습니다. (제출한 파일의 57-60번 라인)

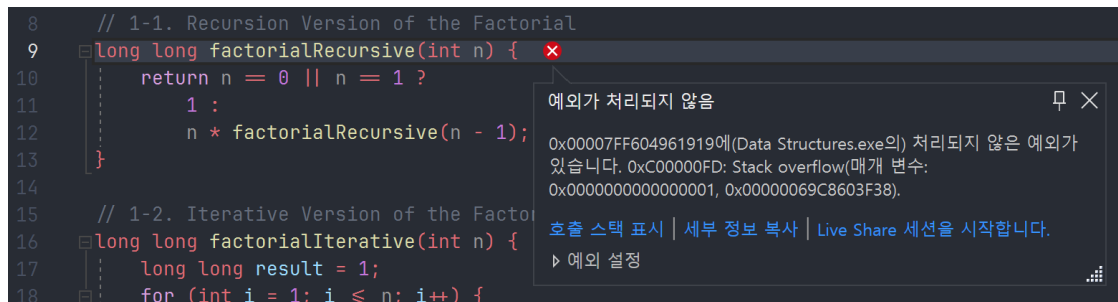
```
1 // 3, 5. Testing Stack Overflow with Recursive Function
2 for (int i = 1; i < 10000; i *= 2) {
3     printf("%d!\t= %lld\n", i, factorialRecursive(i));
4 }
```

위 코드는 `factorialRecursive` 함수의 파라미터  $n$  값을 1부터 시작해 2씩 곱하며 10,000 전 까지의 수(이 경우 8,192까지) 모두를 테스트 하는 코드입니다. 위의 코드를 실행한 결과는 다음과 같습니다.



```
C:\Users\dexrhee\source\repro x + v
1! = 1
2! = 2
4! = 24
8! = 40320
16! = 20922789888000
32! = -6045878379276664832
64! = -9223372036854775808
128! = 0
256! = 0
512! = 0
1024! = 0
2048! = 0
```

Output을 확인해보면,  $n$ 의 값이 작을 경우 값이 정상적으로 표현되나, `factorialRecursive` 함수의 Return type인 **long long**의 범위를 초과하는 수에 대해서는 Overflow가 일어나 정상적인 값이 표현되지 않는 것을 볼 수 있습니다. 또한, `factorialRecursive(2048)`의 값까지는 정상적으로 출력되지만, 이후의 값은 나타나지 않는 모습을 확인할 수 있습니다.



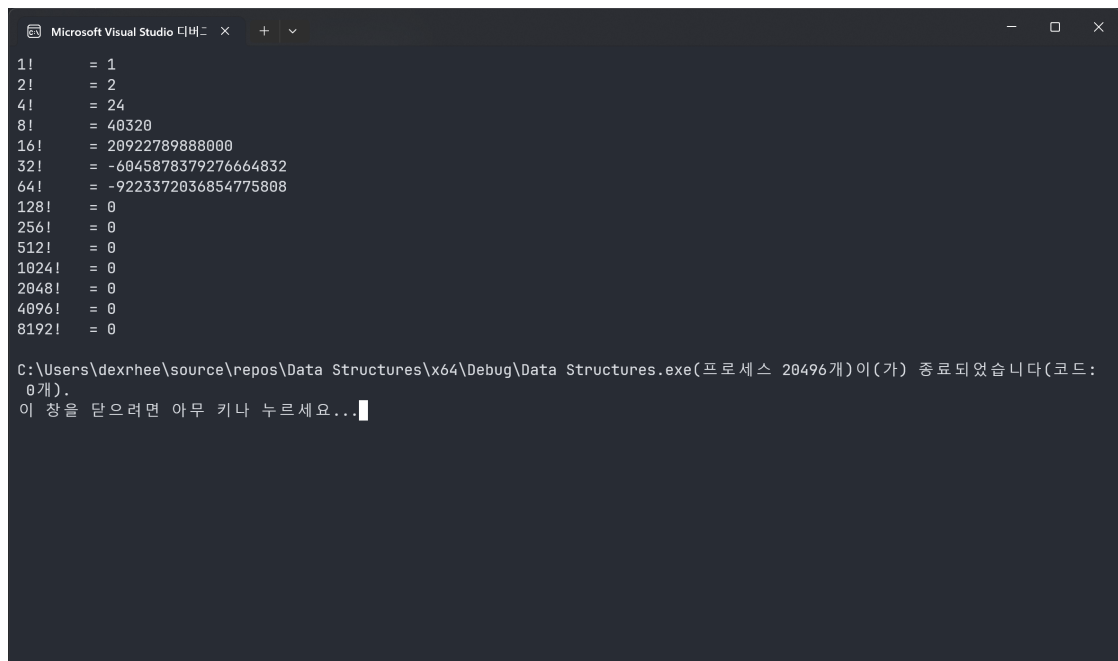
Visual Studio의 Debug 도구를 사용하여 왜 2048 이후의 값이 출력되지 않았는지 확인해보면, Stack Overflow가 발생하여 run time error로 수행이 멈추었다는 것을 확인할 수 있습니다.

**Problem 4** 하지만, 이 경우( $n = 5000$ )에도 iteration version은 실행이 잘 됨을 보이시오. 이 경우도 결과 값은 틀려도 됩니다. (의도: loop이 끝까지 동작해서, 만약 결과값만 잘 답을 수 있는 방법이 있다면 실행이 가능함을 확인하는 것입니다.)

풀이 위 상황을 테스트하기 위해 사용한 코드는 아래와 같습니다. (제출한 파일의 62-65번 라인)

```
1 for (int i = 1; i < 10000; i *= 2) {
2     printf("%d!\t= %lld\n", i, factorialIterative(i));
3 }
```

위 코드는 factorialIterative 함수의 파라미터  $n$  값을 1부터 시작해 2씩 곱하며 10,000 전까지의 수(이 경우 8,192까지) 모두를 테스트 하는 코드입니다. 위의 코드를 실행한 결과는 다음과 같습니다.



`factorialRecursive` 함수의 Return type인 **long long**의 범위를 초과하는 수에 대해서는 Overflow가 일어나 정상적인 값이 표현되지 않는 것을 볼 수 있습니다. 또한, Recursion version과는 다르게, `factorialIterative(2048)` 이후의 값도 정상적으로 출력되는 것을 확인할 수 있습니다.

### Problem 5

(1) 프로그램이 프로세스로서 실행될 때, 구성되는 메모리 공간에 대해서 조사하시오. (즉, 코드영역, Data 영역, Heap 영역, Stack 영역)

*Solution.* 프로그램이 실행되기 위해서는, OS가 프로그램의 정보를 메모리에 로드해야 합니다. 또한 프로그램이 실행되는 동안 CPU가 코드를 처리하기 위해 메모리가 명령어와 데이터들을 저장해야 합니다. 이렇게 프로그램이 OS로 부터 할당받는 대표적인 메모리 공간은 Code 영역, Data 영역, Heap 영역, Stack 영역으로 구성되어 있습니다.

Code 영역은 실행할 프로그램의 코드가 저장되는 영역으로, CPU는 코드 영역에 저장된 명령을 하나씩 가져가 처리합니다. 저장된 프로그램의 코드는 프로그램이 시작하고 종료될 때까지 메모리에 계속 남아있습니다.

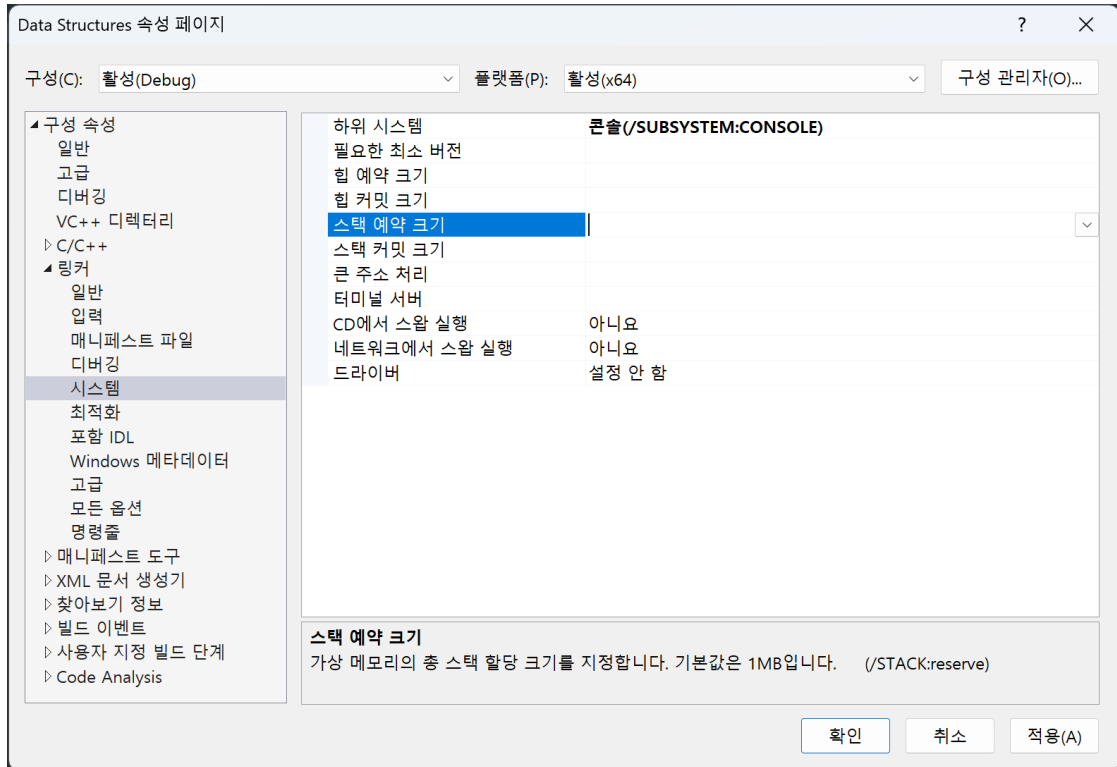
Data 영역은 프로그램의 Global 변수와 Static 변수가 저장되는 영역으로, 프로그램의 시작과 함께 할당되며 프로그램이 종료되면 소멸합니다.

Heap 영역은 프로그래머가 직접 공간을 할당, 해제하는 메모리 공간입니다. Heap 영역에서 공간을 할당, 해제받을 수 있는 C언어의 대표적인 방법으로는, `malloc()`, `free()`가 있습니다. Heap 영역은 메모리의 낮은 주소에서 높은 주소의 방향으로 할당되기 때문에, FIFO<sup>First-In First-Out</sup> 구조입니다.

Stack 영역은 프로그램이 자동으로 사용하는 임시 메모리 영역입니다. 함수 호출 시 생성되는 지역 변수와 매개 변수가 저장되는 영역이고, 함수 호출이 완료되면 사라집니다. 이때 Stack 영역에 저장되는 함수의 호출 정보를 Stack Frame 이라고 합니다. Stack 영역은 메모리의 높은 주소에서 낮은 주소의 방향으로 할당 되기 때문에, LIFO<sup>Last-In First-Out</sup> 구조입니다.

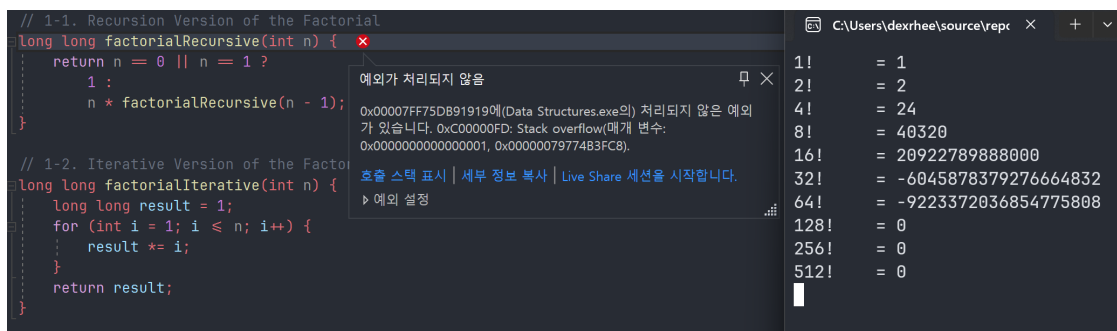
(2) 그리고, Visual studio (또는 본인이 사용 하는 개발 환경)에서 Stack 영역의 크기를 조정하여, 위 3번 문제의 실행될 수 있는 한계가 달라짐을 보이시오.

**Solution.** Visual Studio에서 Stack의 크기를 조정하려면, 프로젝트 > 속성 > 링커 > 시스템에서 스택 예약 크기를 변경하면됩니다.



Visual Studio에서의 기본 Stack Size는 1MB( $2^{20}B = 1,048,576B$ )이므로, 이와 다른 값을 적절히 지정 해주면 위 3번 문제의 실행에 유의미한 변화를 볼 수 있을 것입니다.

**Case 1.** Stack Size를 1KB = 1,024B로 설정한 경우



위 3번에서 실행한 코드와 동일한 코드를 1,024B의 Stack Size를 가지고 실행했을 경우, 위의 그림에서 볼 수 있듯 factorialRecursive(512)까지는 정상 실행되지만, factorialRecursive(1024)부터 Stack Overflow가 발생하는 것을 볼 수 있습니다.



Case 2. Stack Size를 10MB = 10,485,760B로 설정한 경우

```

Microsoft Visual Studio 디버깅
1! = 1
2! = 2
4! = 24
8! = 40320
16! = 20922789888000
32! = -6045878379276664832
64! = -9223372036854775808
128! = 0
256! = 0
512! = 0
1024! = 0
2048! = 0
4096! = 0
8192! = 0

C:\Users\dexrhee\source\repos\Data Structures\x64\Debug\Data Structures.exe(프로세스 20728개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

위 3번에서 실행한 코드와 동일한 코드를 10MB의 Stack Size를 가지고 실행했을 경우, 위의 그림에서 볼 수 있듯 factorialRecursive(8192)까지 모두 실행되고, 정상적으로 프로그램이 종료되었음(Exit code가 0임)을 확인할 수 있습니다.

(3) Visual studio에는 컴파일 옵션이 디버그모드와 릴리즈모드가 있는데, 그 차이점을 조사하고, 위 3번 문제의 실험 결과가 두 가지 모드에서 다르게 나오는 것을 보이시오.

*Solution.* Visual Studio에서 프로젝트를 Build 하는 방법에는 Release와 Debug의 두 가지 방식이 있습니다. Release 모드와 Debug 모드는 컴파일러 최적화, 코드 실행 중 예외 처리, 코드의 안정성, 코드의 크기 면에서 차이가 있습니다.

먼저 컴파일러 최적화 면에서, Debug 모드는 최적화를 거의 하지 않아서 디버깅 시 변수의 값을 추적하기 쉽습니다. 반면에 Release 모드는 최적화를 많이 하기 때문에 실행 속도가 빨라집니다. 하지만 디버깅 시 변수 추적이 어렵습니다.

코드 실행 중 예외 처리 면에서, Debug 모드는 발생한 예외를 즉시 보고하고, 디버거를 통해 프로그램의 상태를 쉽게 파악할 수 있습니다. 반면에 Release 모드는 발생한 예외를 적절히 처리하고, 프로그램이 멈추는 것을 방지합니다. 하지만 예외가 발생한 위치를 찾기 어렵습니다.

코드의 안정성 면에서, Debug 모드는 코드의 안정성을 위해 추가적인 검사와 확인을 수행합니다. 반면에 Release 모드는 실행 속도를 위해 검사를 줄이기 때문에, 더 적은 검사를 수행하고 실행 속도를 빠르게 합니다.

코드의 크기 면에서, Debug 모드는 디버깅을 위해 많은 정보를 포함하고 있기 때문에 코드 크기가 큼니다. 반면에 Release 모드는 최적화를 많이 하기 때문에 코드 크기가 작아집니다.

요약하면, Debug 모드는 디버깅과 코드 안정성에 중점을 두고, Release 모드는 실행 속도와 코드 크기에 중점을 두는 것입니다. 따라서, 개발 시에는 Debug 모드로 작업하고, 배포 시에는 Release 모드로 컴파일하여 실행하는 것이 적합합니다.

위의 내용을 참고하여 두 가지 모드에 따라 실행 결과가 달라지는지 확인해보도록 합시다. 이전에 보았듯, Debug 모드에서 위 3번의 코드를 실행하였을 때에는 다음의 그림과 같이 `factorialRecursive(2048)`까지만 작동하고 `factorialRecursive(4096)`부터는 Stack Overflow가 발생함을 알 수 있습니다.

The screenshot shows a Windows command prompt window with the following output:

```

1! = 1
2! = 2
4! = 24
8! = 40320
16! = 20922789888000
32! = -6045878379276664832
64! = -9223372036854775808
128! = 0
256! = 0
512! = 0
1024! = 0
2048! = 0

```

Below the command prompt, the Visual Studio code editor shows the following C++ code:

```

8 // 1-1. Recursion Version of the Factorial
9 long long factorialRecursive(int n) {
10     return n == 0 || n == 1 ?
11         1 :
12         n * factorialRecursive(n - 1);
13 }
14
15 // 1-2. Iterative Version of the Factorial
16 long long factorialIterative(int n) {
17     long long result = 1;
18     for (int i = 1; i <= n; i++) {

```

An error dialog box is displayed over the code, indicating a stack overflow. The text in the dialog box is as follows:

예외가 처리되지 않음

0x00007FF604961919에(Data Structures.exe의) 처리되지 않은 예외가 있습니다. 0xC00000FD: Stack overflow(매개 변수: 0x0000000000000001, 0x00000069C8603F38).

호출 스택 표시 | 세부 정보 복사 | Live Share 세션을 시작합니다.

예외 설정

하지만 Release 모드로 프로젝트를 Build 후 실행하였을 때에는 다음의 그림과 같이 Stack Overflow가 발생하지 않고 `factorialRecursive(8192)`의 값까지 모두 출력되고 정상적으로 프로그램이 종료되었음(Exit code가 0임)을 확인할 수 있습니다.

```

Microsoft Visual Studio 디버깅
1! = 1
2! = 2
4! = 24
8! = 40320
16! = 20922789888000
32! = -6045878379276664832
64! = -9223372036854775808
128! = 0
256! = 0
512! = 0
1024! = 0
2048! = 0
4096! = 0
8192! = 0

C:\Users\dexrhee\source\repos\Data Structures\x64\Release\Data Structures.exe(프로세스 6124개)이(가) 종료되었습니다(코드 : 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

따라서 Release 모드로 프로젝트를 Build 하였을 때, 내부적으로 최적화가 이루어져 Stack Overflow가 일어나지 않음을 알 수 있겠습니다.

### 실험 환경

본 실험은 다음과 같은 환경에서 진행되었으며, 실행을 진행하는 환경에 따라 위의 내용과 상이할 수 있습니다.

프로세서	Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69GHz
RAM	16.0GB
OS	Windows 11 22H2(22621.1265)
IDE	Microsoft Visual Studio Community 2022 (64-bit) v17.5.2

### 참고 문헌

1. all-young, 메모리의 구조 (코드, 데이터, 힙, 스택 영역), 티스토리 블로그, 2020년 9월 4일 작성, 2023년 3월 20일 조회 (<https://all-young.tistory.com/17>)
2. 코딩팩토리, [IDE] 비주얼 스튜디오 Release, Debug 모드의 차이점, 티스토리 블로그, 2021년 1월 24일 작성, 2023년 3월 20일 조회 (<https://coding-factory.tistory.com/648>)