Object-Oriented Programming Lab #07

Department: 응용물리학과 Student ID: 2017103038 Name: 권인회

A. Code explanation & output analysis (Write the source code and results)

A-1. Listing 10.9

< 코드 >

```
#include <iostream>
// Draws a bar n segments long
// using iteration.
void segments1(int n) {
        while (n > 0) {
                std::cout << "*";
                n--;
        }
        std::cout << '₩n';
}
// Draws a bar n segments long
// using recursion.
void segments2(int n) {
        if (n > 0) {
                std::cout << "*";
                segments2(n - 1);
        }
        else
                std::cout << '₩n';
}
int main() {
        segments1(3);
        segments1(10);
        segments1(0);
        segments1(5);
        std::cout << "----------₩n";
        segments2(3);
        segments2(10);
        segments2(0);
        segments2(5);
}
```

< 실행결과 >

```
***

**********



*****
```

***

****

*****

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 131712개)이(

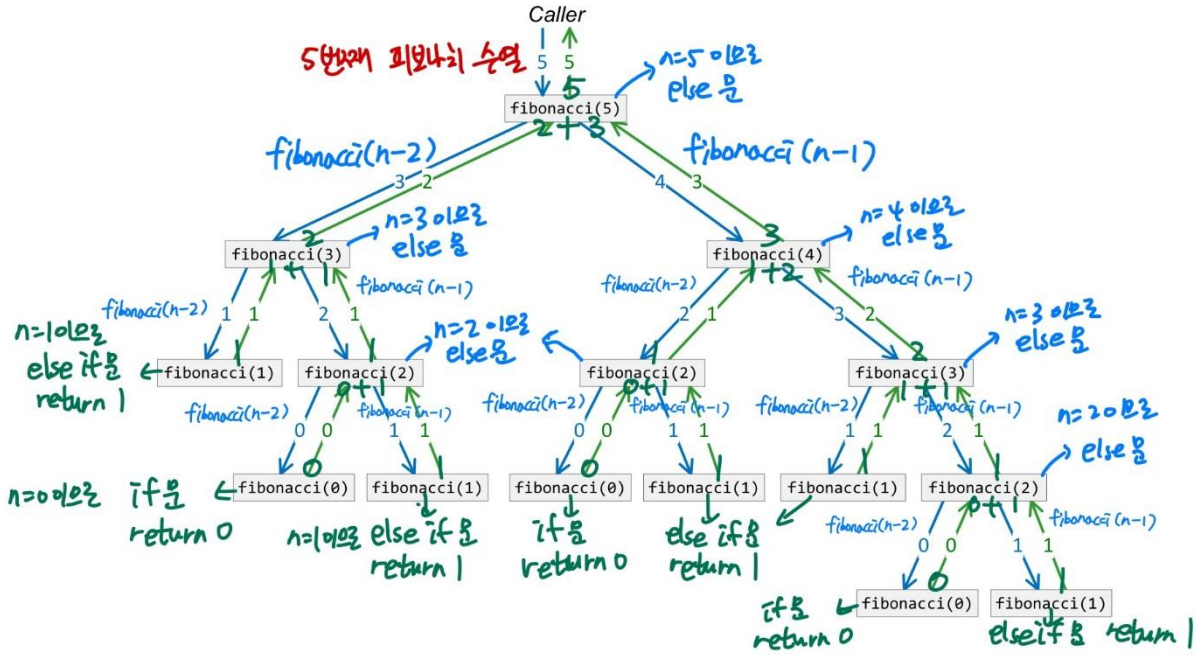
가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

함수가 2개가 정의되어있다. Segments1과 달리 segments2는 recursion이다. 하지만 두 함수는 같은 기능을 수행한다. Parameter로 받은 int n의 수만큼 *을 출력하는 함수이다. Segments1 함수는 이전에 많이 살펴봤으므로 segments2 함수를 살펴보자. if문에 n이 0보다 큰 경우에 들어간다. 1 이상일 경우에 if문 안으로 들어가므로 우선 *을 출력하고 n-1을 parameter로 하는 segments2 함수를 호출한다. 그렇다면 처음 실행한 함수 segments2는 if문 안에서 멈춘 상태로 또 다시 segments2 함수로 들어가는 것이다. 만약, n-1이 또 0보다 크다면 if문 안에 들어가서 *을 출력하고 n-2를 parameter로 하는 segments2 함수를 호출한다. 만약, 이제 이 n-2가 0보다 크지 않다면 else문에 들어가 줄바꿈을 하고 연쇄적으로 함수에서 나오면서 제일 처음 들어왔던 함수의 if문에서 빠져나오면서 함수 실행이 끝난다. 지금 설명한대로라면 n=2일 때, *이 2개 출력된것이다. 따라서, segments1 함수와 동일하고 정상적으로 함수 기능을 수행하는 recursion이다.

A-2. Figure 10.2 (Explain the process of recursion when Fibonacci(5) is called)



Fibonacci 함수에 따르면 파라미터를 int형 변수 n으로 받아 n이 0보다 작거나 같으면 0을 return, n이 1이면 1을 return, 그 외에는 Fibonacci(n-2) + Fibonacci(n-1)을 return 하는 resursion이다. 따라서, 아래서부터 차곡차곡 더해 올라와서 피보나치 수열의 n번째 수를 return해 내는 것이다. 자세한 설명은 그림을 참고해주세요.

B. Exercises (Write the questions down on your answer sheet)

(pp. 280-287), Exercises 1-10, 12
(write output analysis for all exercises)

1. Consider the following C++ code:

```
#include <iostream>
int sum1(int n) {
        int s = 0;
        while (n > 0) {
                s++;
                n--;
        }
        return s;
}
int input;
int sum2() {
        int s = 0;
        while (input > 0) {
```

```
                s++;
                input--;
        }
        return s;
}
int sum3() {
        int s = 0;
        for (int i = input; i > 0; i--)
                s++;
        return s;
}
int main() {
        // See each question below for details
}
```

(a) What is printed if main is written as follows?

```
int main() {
        input = 5;
        std::cout << sum1(input) << '₩n';
        std::cout << sum2() << '₩n';
        std::cout << sum3() << '₩n';
}
```

< 실행결과 >

5

5

0

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 137948개)이(

가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

Sum1함수의 경우 1개의 parameter를 필요로 하기 때문에 input이라는 변수를 인수로 넣어주었다. Main 내에서 input 변수는 5로 설정되어있으므로 인수로 5가 들어가서 sum1의 함수를 실행한다. while문이 도는 동안 5가 1씩 빠지고 0으로 초기화되있는 변수 s가 1씩 더해지면서 while문이 끝나면 s가 5가 되어 return 된다. Sum2함수의 경우 인수가 필요 없다. 하지만 기본적으로 함수 내에서 input변수를 선언하지 않은 채 사용하기 때문에 main 내에서 들어가있는 input 변수를 사용한다. 따라서, 5의 값이 input으로 들어가서 함수를 실행하여 sum1과 같이 s는 5가 return된다. Sum3는 함수 내에서 input 변수를 다른 변수인 i에 할당한다. 하지만 input 변수는 sum2 함수 내에서 값이 0이 되었다. 따라서 sum3가 받은 변수 input의 값은 0이다. 그렇게 sum3에서는

input이 0이기 때문에 for문에 들어가지 않고 바로 s=0이 return 된다.

(b) What is printed if main is written as follows?

```
int main() {
        input = 5;
        std::cout << sum1(input) << '₩n';
        std::cout << sum3() << '₩n';
        std::cout << sum2() << '₩n';
}
```

< 실행결과 >

5

5

5

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 106804개)이(

가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

앞선 (a)와 sum2 함수와 sum3 함수의 위치만 변경되었다. 따라서 앞선 설명과 동일하나, sum3 함수에서는 input 변수의 값을 조절하지 않았으므로 그대로 5가 유지되었기 때문에 뒤에 있는 sum2에도 input 변수의 값은 5가 들어가기 때문에 모든 함수가 5로 출력된다.

(c) What is printed if main is written as follows?

```
int main() {
        input = 5;
        std::cout << sum2() << '₩n';
        std::cout << sum1(input) << '₩n';
        std::cout << sum3() << '₩n';
}
```

< 실행결과 >

5

0

0

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 131008개)이(

가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

앞선 문제에서 계속 말했던 것처럼 sum2 함수는 안에서 input 변수를 계속 차감시켜 결국엔 0이 된다. 따라서, sum2 함수에서는 정상적으로 s의 값인 5가 return 되어 출력되지만, 뒤에 있는 sum1, sum3함수에는 input 변수의 값이 main 내에서 선언된 5가 아니라 sum2에서 5였던 값이 0이 되어 나왔기 때문에 이 0이라는 값이 들어가서 s가 0으로 return 되고 출력되는 것이다.

(d) Which of the functions sum1, sum2, and sum3 produce a side effect? What is the side effect?

: sum2이다. 변수 input의 값을 변화시켜 뒤에 오는 코드에 영향을 준다.

(e) Which function may not use the input variable?

: sum1 함수가 input 변수를 사용하지 않는다. 하지만 인수를 필요로 하는데 문제에서는 그 인수에 input 변수가 들어갔기 때문에 영향을 받은 것이다.

(f) What is the scope of the variable input? What is its lifetime?

: input 변수가 전역변수로 선언되었기 때문에 모든 곳에서 유효하다. 그래서 모든 곳에서 영향을 받는 것이다.

(g) What is the scope of the variable i? What is its lifetime?

: I 변수는 sum3 함수 내에서만 유효하다. Sum3 함수가 return 될 때, 변수 i도 수명을 다한다.

(h) Which of the functions sum1, sum2, and sum3 manifest good functional independence? Why?

: 문제가 무슨뜻인지 잘 모르겠지만... 아마 sum1 함수가 parameter를 받기 때문에 다른 변수들에

의존받지 않고 독립적으로 기능 수행을 잘 하지 않나 싶다. 모든 변수가 sum1 내에서 선언되었기때문에 그렇다.

2. Consider the following C++ code:

```
#include <iostream>
int next_int1() {
        static int cnt = 0;
        cnt++;
        return cnt;
}
int next_int2() {
        int cnt = 0;
        cnt++;
        return cnt;
}
int global_count = 0;
int next_int3() {
        global_count++;
        return global_count;
}
int main() {
        for (int i = 0; i < 5; i++)
                std::cout << next_int1() << " "
                << next_int2() << " "
                << next_int3() << '₩n';
}
```

(a) What does the program print?

< 실행결과 >

```
1 1 1

2 1 2

3 1 3

4 1 4

5 1 5
```

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 153312개)이(

가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

Next_int1 함수에서는 cnt를 정적변수로 초기화하였다. 그렇기 때문에 한번 선언하면 그 변수를 계속 가지고 가기 때문에 함수 안에서 초기화를 했더라도 계속 유효하다. 하지만 함수 밖에서는 접근할 수 없다. 따라서 이 함수를 한번 실행시켰을 때 cnt++가 되어 이 cnt 변수값은 1이 되고 계속 유효한 것이다. 따라서 두번째 실행시에는 2가 되고 점점 올라가는 것이다. Next_int2 함수는 함수 안에서만 사용하는 지역변수 cnt이므로 실행할때마다 cnt 값이 0으로 초기화 된 뒤 사용되는거라 계속 1인 cnt값만 return 되는 것이다. Next_int3 함수는 전역변수로 선언된 변수를 사용하기 때문에 next_int1 함수와 비슷한 원리로 계속 1씩 누적되어 출력되는 것이다.

(b) Which of the functions next_int1, next_int2, and next_int3 is the best function for the intended purpose? Why?

: next_int1 함수이다. 왜냐하면 실행결과에 나와있다시피 next_int2 함수는 의도된 기능을 수행하지 못하고, next_int3 함수는 전역변수를 사용하기 때문에 다른 범위에서 그 변수에 다른 값을 할당하면 값이 바뀐다. 하지만 next_int1 함수에서는 다른 범위에서는 접근이 불가하지만 계속 유효한 변수인 정적변수를 사용하기 때문에 의도된 목적을 그대로 수행 가능하다.

(c) What is a better name for the function named next_int2?

: 본인 맘대로 바꾼다면 one 이라는 변수를 쓰겠다. 무조건 1이 나오기 때문이다.

(d) The next_int3 function works in this context, but why is it not a good implementation of a function that always returns the next largest integer?

: 아까 말했다시피 다른 범위에서 그 변수의 값을 수정할 수 있어서 실행할때마다 차례대로 다음 순서의 정수가 나오지 않을 수도 있다.

3. The following C++ program is split up over three source files. The first file, counter.h, consists of

```
int read();
int increment();
int decrement();
```

The second file, counter.cpp, contains

```
static int count;
int read() {
        return count;
}
```

```
int increment() {
        if (count < 5)
                count++;
}
int decrement() {
        if (count > 0)
                count--;
}
```

The third file, main.cpp, is incomplete:

```
#include <iostream>
#include "counter.h"
int main() {
        // Add code here
}
```

(a) Add statements to main that enable it to produce the following output:

3

2

4

The restriction is that the only ouput statement you are allowed to use (three times) is

std::cout << read() << '₩n';

< 코드 >

```
#include <iostream>
#include "counter.h"
int main() {
        increment();
        increment();
        increment();
        std::cout << read() << '₩n';
        decrement();
        std::cout << read() << '₩n';
        increment();
        increment();
        std::cout << read() << '₩n';
}
```

< 실행결과 >

3

2

4

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 153464개)이(

가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

Couter.cpp 파일에 함수가 정의되어있다. 여기서 살펴보면 이 파일 전체에서만 접근 가능한 count변수가 정적변수로 선언되어있다. 이 파일 안에서만 접근 가능하고, 유효하기 때문에 여기에 있는 함수들에 의해 값이 한번 변하면 계속 유효하다. 따라서, main.cpp 파일에서 counter.cpp에 있는 함수를 호출하여 count 값을 변동시키고 그에 따른 값을 출력하면 된다. 따라서, increment 함수를 3회하여 count 값을 3으로 올리고, read 함수를 출력하여 count 변수의 값을 출력하고, decrement 함수를 1회 하여 count 값을 2로 내리고 read함수 출력, 다시 increment 함수를 2회 하여 count 값을 4로 올리고 read 함수 출력. 따라서 결과물을 만들어낸다. 이것이 정적변수 count의 값이 계속 유효하게 남아있기 때문에 가능하다.

(b) Under the restriction of using the same output statement above, what code could you add to main so that it would produce the following output?

6

: 주어진 함수와 제약사항으로는 불가능한 것 같습니다. Increment 함수에서 count >= 5인 경우에는 count++가 되지 않아 최대 5까지는 표현이 가능한데... (교수님 채점 X)

4. Consider the following C++ code:

```
#include <iostream>
int max(int n) {
        return n;
}
int max(int m, int n) {
        return (m >= n) ? m : n;
}
int max(int m, int n, int r) {
        int x = m;
        if (n > x)
                x = n;
        if (r > x)
                x = r;
        return x;
}
```

```
int main() {
	std::cout << max(4) << '₩n';
	std::cout << max(4, 5) << '₩n';
	std::cout << max(5, 4) << '₩n';
	std::cout << max(1, 2, 3) << '₩n';
	std::cout << max(2, 1, 3) << '₩n';
	std::cout << max(2, 1, 2) << '₩n';
}
```

(a) Is the program legal since there are three different functions named max?

Legal, max 함수가 1,2,3개의 변수를 받았을 때 전부 실행할 수 있도록 오버로딩해둔 것이다. 절차상 문제는 없으며, 변수의 개수에 따라 각 함수가 실행되어 가장 큰 수를 return 한다.

(b) What does the program print?

4

5

5

3

3

2

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 62220개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

: 가장 큰 값이 원활하게 출력되는 것을 볼 수 있다.

5. Consider the following function:

```
#include <iostream>
int proc(int n) {
	if (n < 1)
		return 1;
	else
		return proc(n / 2) + proc(n - 1);
}
```

Evaluate each of the following expressions:

(a) proc(0)

: 1이 return된다. N이 1보다 작기 때문에 if문이 실행되어 return 1.

(b) proc(1)

: 2가 return된다. else문에 들어가서 proc(1/2)+proc(0)을 실행하여 1+1이 되어 return 2.

Proc(1/2)는 n이 int형이기 때문에 proc(0) 취급되어 실행된다.

(c) proc(2)

: 4가 return된다. else문에 들어가서 proc(1)+proc(1)을 실행하여 2+2가 되어 return 4

(d) proc(3)

: 6이 return된다. else문에 들어가서 proc(3/2)+proc(2)를 실행하여 2+4가 되어 return 6.

Proc(3/2)는 n이 int형이기 때문에 proc(1) 취급되어 실행된다.

(e) proc(5)

: 14가 return된다. else문에 들어가서 proc(5/2)+proc(4)를 실행하여 proc(2)+{proc(2)+proc(3)}이 되면 위의 답들 중에서 대입하면 4+4+6이 되어 return 14.

(f) proc(10)

: 60이 return된다. else문에 들어가서 proc(5)+proc(9)를 실행하여 14+proc(4)+proc(8)를 실행하여 24+proc(4)+proc(7)을 실행하여 34+proc(3)+proc(6)을 실행하여 40+proc(3)+proc(5)를 실행하면 40+6+14이므로 return 60. (대입값은 전부 위의 답안에서 사용)

(g) proc(-10)

: 1이 return된다. N이 1보다 작으면 if문에 들어가서 return 1.

6. Rewrite the gcd function so that it implements Euclid's method but uses iteration instead of recursion.

< 코드 >

```
#include <iostream>
int gcd(int a, int b) {

    if (b == 0) {
        return a;
    }
    else {
        return gcd(b, a % b);
```

```
    }
}

int main() {
    std::cout << gcd(18, 24);
}
```

< 실행결과 >

6

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 68268개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

Recursion 이용해서 gcd 함수를 작성하였다. 두 수의 최대공약수를 구하는 것이며 두 수를 파라미터로 받는다. 이 때, recursion 할 때 a를 b로 나눈 나머지를 이용할 것이기에 b의 값이 0이면 안되므로 b의 값이 0인 경우에 a를 return하도록 하였다. 만약, a가 0인 경우에도 바로 다음 recursion에서 b가 return되도록 되어있다. 예시를 들어 설명하겠다. A에 18, b에 24를 넣은 경우 else문으로 들어가서 다음 gcd에서는 (24, 18)을 파라미터로 가져간다. 여기서도 else문으로 들어가서 다음 gcd에서는 (18,6)을 파라미터로 가져가고 여기서도 else문으로 들어가서 다음 gcd에서는 (6,0)을 가져가고 여기서는 if문으로 들어가 return a, 즉 6이 return 된다. 따라서 6이 return 된다.

7. If x is a variable, how would you determine its address in the computer's memory?

: 컴퓨터 메모리의 각 바이트에는 고유한 주소가 번호로 지정된다. 첫번째 주소는 0이고 위치 번호는 운영체제와 하드웨어에서 허용하는 최대값까지 순차적으로 지정된다. 변수는 메모리에 저장되므로 각 변수는 특정 주소에 상주한다. 간단하게 말하자면 변수 x의 기억부류에 따라 알맞게 할당되어 저장된다.

8. What is printed by the following code fragment?

```
#include <iostream>

int main() {
    int x = 5, y = 3, * p = &x, * q = &y;
    std::cout << "x = " << x << ", y = " << y << '₩n';
    x = y;
    std::cout << "x = " << x << ", y = " << y << '₩n';
```

```
    x = 7;
    std::cout << "x = " << x << ", y = " << y << '₩n';
    *p = 10;
    std::cout << "x = " << x << ", y = " << y << '₩n';
    p = q;
    *p = 20;
    std::cout << "x = " << x << ", y = " << y << '₩n';
}
```

< 실행결과 >

x = 5, y = 3

x = 3, y = 3

x = 7, y = 3

x = 10, y = 3

x = 10, y = 20

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 71728개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

5번의 출력을 순차적으로 설명한다. 1번째는 그냥 들어간 변수를 출력한다. 2번째는 x에 y값을 할당하였으므로 둘 다 3이 나온다. 세번째는 x에 7을 새로 할당해줬으므로 당연히 7과 3이 나온다. 네번째를 보자. 변수 x의 주소에 접근해있는 포인터값을 10으로 바꿔주었다. 그래서 변수 x의 값이 10으로 바뀌어 10과 3이 나온다. 다섯번째는 p에 q를 할당하였다. Q는 y의 주소에 접근해있는 포인터이므로 이 역할을 p가 하게 되는 것이다. 그래서 이후에 p에 20을 할당했을 때 원래 q의 역할이었던 변수 y의 주소에 접근해있는 포인터값을 20으로 바꿔 x는 그대로 10, y는 20이 출력되게 하는 것이다.

9. Given the declarations: int x, y, *p, *q; indicate what each of the following code fragments will print.

(a) p = &x; x = 5; std::cout << *p << '₩n';

: 5가 출력된다. P는 x의 주소를 가리키는 포인터인데, x의 값에 5가 할당되어있으므로 p에는 x의 주소가 들어있으므로 *p로 주소에 참조된 값을 출력하면 그것이 변수 x의 값이기에 당연히 5가

출력된다.

(b) x = 5; p = &x; std::cout << *p << '₩n';

: 5가 출력된다. 위와 코드 순서만 바뀐것인데, 변수 x에 먼저 5를 할당해주고 p를 x의 포인터로 할당해준 것이다. 그러면 어차피 p는 x의 주소값이므로 주소에 참조된 값을 출력하면 x의 값인 5가 출력되는 것이다.

(c) p = &x; *p = 8; std::cout << *p << '₩n';

: 8이 출력된다. P가 x의 주소값을 가리키는 포인터, 이 p에 참조된 값에 8을 할당, 그렇다면 x의 값에도 8이 들어감과 동시에 p에 참조된 값을 출력하면 당연히 x의 주소에 참조된 값이 나오므로 8이 출력된다.

(d) p = &x; q = &y; x = 100; y = 200; *q = *p; std::cout << x << ' ' << y << '₩n'; std::cout << *p << ' ' << *q << '₩n';

< 실행결과 >

100 100

100 100

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 68452개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

P는 x의 주소를 가리키는 포인터, q는 y의 주소를 가리키는 포인터, 여기서 x에 100, y에 200을 할당하고 q의 참조값에 p의 참조값을 대입한다. 그러면 변수 y에 x의 값을 대입하는 효과를 불러온다. 따라서 x,y 모두 100의 값을 가지며, 이를 가리키는 포인터의 참조값 또한 100인것이다.

(e) p = &x; q = &y; x = 100; y = 200; q = p; std::cout << x << ' ' << y << '₩n'; std::cout << *p << ' ' << *q << '₩n';

< 실행결과 >

100 200

100 100

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 76300개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

위와 같은 상황에서 코드 하나만 달라졌다. 포인터인 Q의 참조값에 포인터인 p의 참조값을 넣은 것이 아니라 그냥 q에 p를 대입했다. 따라서, 포인터 q는 포인터 p와 같은 주소를 같게 되어 같은 값을 출력하게 되지만... 그것이 주소에 참조되어 있는 값인 y에게는 영향을 주지 않으므로 y 값은 초기화된 200의 값을 그대로 출력하는 것이다.

(f) x = 5; y = 10; p = q = &y; std::cout << *p << ' ' << *q << '₩n'; *p = 100; *q = 1; std::cout << x << ' ' << y << '₩n';

< 실행결과 >

10 10

5 1

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 73424개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

X에 5, y에 10을 할당해준다. 이후 p와 q가 y의 주소를 가리키는 포인터로 지정해준다. 그래서 p와 q의 참조값을 출력하면 당연히 y에 들어있는 10이 출력된다. 여기서 p에 참조값을 100으로 할당하고 q에 참조된 값을 1로 할당한다. 같은 주소에 대한 포인터를 바꾼 것이므로 y의 값은 100이 되었다가 1이 된다. 따라서, 원래 x에는 아무 연관이 없으므로 처음에 할당된 5가 출력되고 y에는 1이 출력되는 것이다.

(g) x = 5; y = 10; p = q = &x; *p = *q = y; std::cout << x << ' ' << y << '₩n';

< 실행결과 >

10 10

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 78784개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

X와 y에 각각 5와 10을 할당한다. P와 q는 x의 주소를 가리키는 포인터이다. 여기서, p와 q의 참조값을 y로 할당해준다. 그렇다면 x의 값이 y인 10으로 바뀌는 것이다. 왜냐하면 포인터의 참조값을 바꿨기 때문에 그 참조값이던 x가 바뀐 것이다. 따라서, x와 y 모두 10으로 출력된다.

10. The following function does not behave as expected:

```
/*
* (Faulty function)
*
* get_range
* Establishes a range of integers. The lower value must
* be greater than or equal to min, and the upper value
* must be less than or equal to max.
* min is the lowest acceptable lower value.
* max is the highest acceptable upper value.
* lower is assigned the lower limit of the range
* upper is assigned the upper limit of the range
*/
void get_range(int min, int max, int lower, int upper) {
    std::cout << "Please enter a data range within the bounds "
        << min << "..." << max << ": ";
    do { // Loop until acceptable values are provided
        std::cin >> lower >> upper;
        if (lower < min)
            std::cout << lower << " is too low, please try again.₩n";
        if (upper > max)
            std::cout << upper << " is too high, please try again.₩n";
    } while (lower < min || upper > max);
}
```

(a) Modify the function so that it works using pass by reference with pointers

< 코드 >

```
void get_range(int *min, int *max, int *lower, int *upper) {
    std::cout << "Please enter a data range within the bounds "
        << *min << "..." << *max << ": ";
    do { // Loop until acceptable values are provided
```

```
        std::cin >> *lower >> *upper;
        if (*lower < *min)
            std::cout << *lower << " is too low, please try again.₩n";
        if (*upper > *max)
            std::cout << *upper << " is too high, please try again.₩n";
    } while (*lower < *min || *upper > *max);
}
```

< 설명 >

역참조 연산자 사용하여 함수 내 변수 앞에 붙여주고 함수를 호출할 때 파라미터에 주소 연산자를 앞에 붙여주면 기대한대로 함수가 실행된다. 변수값이 제대로 할당된다. 포인터를 사용하면 변수의 메모리 위치에 액세스할 수 있다. 이 덕분에 함수를 사용하여 호출자가 소유한 변수 값을 수정할 수 있다. 실제로 포인터가 있는 pass by reference 값의 복사본을 전달하는 대신 주소의 사본을 전달한다. 그것들이 가리키는 메모리를 재할당한다. 그것들은 동일한 주소이고 그렇기에 메모리에 있는 동일한 위치이다.

(b) Modify the function so that it works using pass by reference with reference parameters.

< 코드 >

```
void get_range(int& min, int& max, int& lower, int& upper) {
    std::cout << "Please enter a data range within the bounds "
        << min << "..." << max << ": ";
    do { // Loop until acceptable values are provided
        std::cin >> lower >> upper;
        if (lower < min)
            std::cout << lower << " is too low, please try again.₩n";
        if (upper > max)
            std::cout << upper << " is too high, please try again.₩n";
    } while (lower < min || upper > max);
}
```

< 설명 >

함수의 파라미터를 받을 때 int 뒤에 주소연산자를 붙이면서 reference parameter를 만들어준다. 함수 파라미터를 reference로 하면 파라미터에 값이 전달될 때 참조자와 같이 작동하게 된다. 참조 파라미터이기 때문에 value를 받을 때 값이 그대로 넘어와서 계산된다.

12. Complete the following function that assigns to its mx and my reference parameters the components of the midpoint of the points (x1, y1) and (x2, y2), represented by the parameters x1, y1, x2, and y2.

```
// Computes the midpoint of the points (x1, y1) and (x2, y2).
// The point (mx, my) represents the midoint.
```

```
void midpoint(double x1, double y1, double x2, double y2,
    double& mx, double& my) {
    // Add your code . . .
}
```

< 코드 >

```
#include <iostream>
// Computes the midpoint of the points (x1, y1) and (x2, y2).
// The point (mx, my) represents the midoint.
void midpoint(double x1, double y1, double x2, double y2,
    double& mx, double& my) {
    mx = (x1 + x2) / 2;
    my = (y1 + y2) / 2;
}

int main() {
    double var1 = 2.2, var2 = 4.4, a = 0, b = 0;
    midpoint(var1, var1, var2, var2, a,b);
    std::cout << a << ", " << b;
}
```

< 실행결과 >

3.3, 3.3

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 86864개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

Reference를 사용하지 않았다면 실행결과는 0, 0이 나왔을 것이다. 왜냐하면 함수가 parameter를 받을 때 a와 b의 값을 parameter로 받지만 함수 내의 변수들과는 다르기 때문에 함수 내에서 선언되어있는 mx와 my의 값은 계산되어진 이후 함수를 나올 때 더 이상 유효하지 않는다. 따라서, a와 b에는 그대로 0의 값이 들어있는 것이다. 하지만 reference를 사용하면 a와 b의 참조자로 들어가기 때문에 이 변수들의 값이 그대로 쓰여서 이 변수에 계산된 값이 그대로 들어간 이후에 계속해서 유효할 수 있도록 하기 때문이다. 따라서, 함수를 통해 계산된 값이 정상적으로 출력된다.

C. Additional exercises (Write the questions down on your answer sheet)

C-1. Code explanation & output analysis (Write the source code and results) – see https://en.wikipedia.org/wiki/Tower_of_Hanoi

```
#include <iostream>
void Hanoi(int m, char start, char middle, char end){
        if(m == 1){
                std::cout << "Move disc " << m << " from " << start << " to " << end
                        << std::endl;
        }
        else{
                Hanoi(m-1, start, end, middle);
                std::cout << "Move disc " << m << " from " << start << " to " << end
                        << std::endl;
                Hanoi(m-1, middle, start, end);
        }
}
int main(){
        int discs = 3;
        Hanoi(discs, 'A', 'B', 'C');
}
```

< 실행결과 >

Move disc 1 from A to C

Move disc 2 from A to B

Move disc 1 from C to B

Move disc 3 from A to C

Move disc 1 from B to A

Move disc 2 from B to C

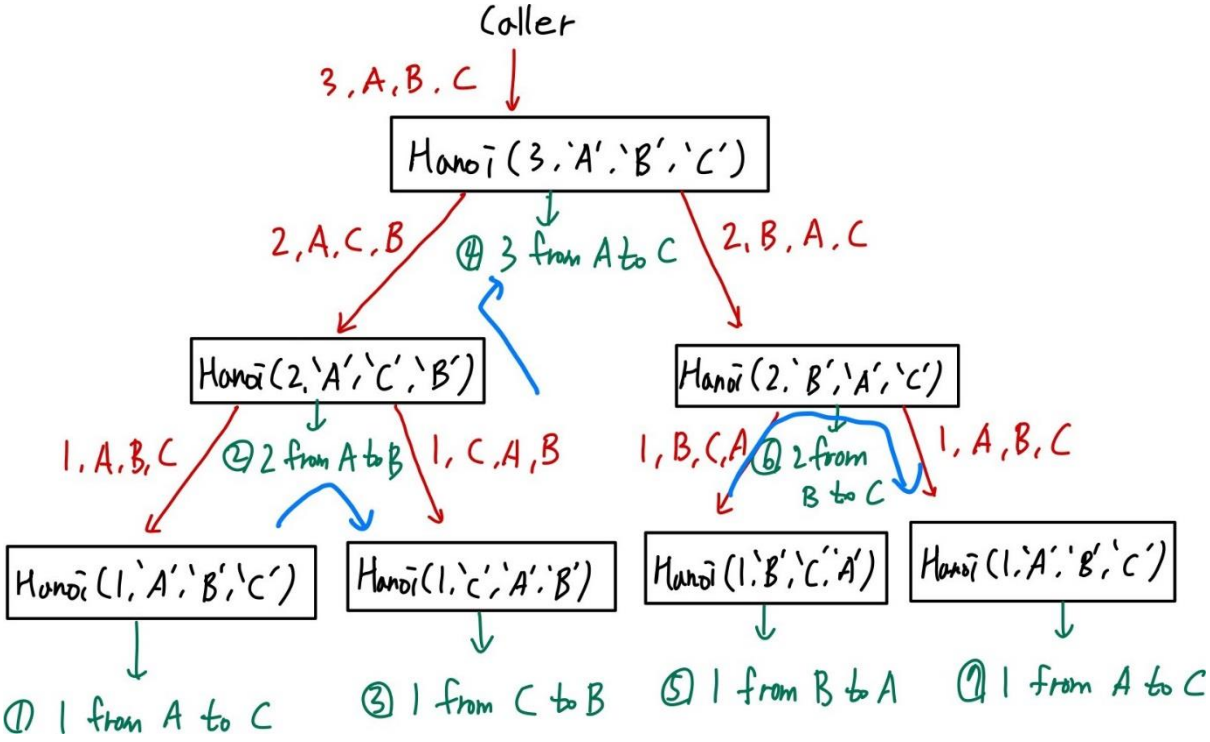
Move disc 1 from A to C

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 92168개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >



순서가 생각보다 복잡하다. 일단, 원판의 개수를 정하고, 시작 막대와 중간 막대, 끝 막대도 정해준다. (구분을 하기 위함) 차례대로 A,B,C 막대라고 할 때, 3개의 원판을 A에서 C로 옮기는 것이 목표이다. 함수에 있는 if, else문을 보면 그림상에서 왼쪽으로 가는 것이 우선인 것을 알 수 있다. 왜냐하면, 처음에는 변수 m이 1보다 높기 때문에 else문으로 들어가는데, else문 안에서 recursion이 두개 있는 것을 볼 수 있다. 하지만 중간에 출력이 한번 발생하기 때문에, 첫번째 recursion을 왼쪽으로 그려두었다. 따라서, 첫번째 recursion에서의 첫번째 parameter가 1이 될때까지 else문을 반복하다가, if문으로 들어갔을 때 첫번째 출력이 일어난다. 그것이 그림에서 표기한 1번이다. 이후 이전 단계로 돌아와 else문의 가운데에 있는 출력을 수행한다. 이후 다음에 있는 recursion에 들어가 if문을 수행한다. 이것이 2번, 3번 단계이다. 이것이 처음 들어간 함수에서 else문에 있는 첫번째 recursion을 수행한 결과이다. 그래서 가운데 끼어있는 출력문을 수행한다. 이것이 4번이다. 이후에 위치한 recursion을 실행하여 방금까지 설명한 단계를 한번 더 반복한다. 그래서 5,6,7번 과정이 실행되고 다시 첫번째 진입한 함수로 돌아와 else문을 벗어나게 되면서 함수가 끝이난다.