Object-Oriented Programming Lab #11

Department: 응용물리학과 Student ID: 2017103038 Name: 권인회

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

A-1. Listing 15.7, 15.8, 15.9

< 15.7 >

```
enum class SignalColor { Red, Green, Yellow };
class Trafficlight {
private:
        SignalColor color; // The light's current color: Red, Green, or Yellow
public:
        Trafficlight(SignalColor initial_color);
        void change();
        SignalColor get_color() const;
};
```

< 15.8 >

```
#include "trafficlight.h"
// Ensures a traffic light object is in the state of
// red, green, or yellow. A rogue value makes the
// traffic light red
Trafficlight::Trafficlight(SignalColor initial_color) {
        switch (initial_color) {
        case SignalColor::Red:
        case SignalColor::Green:
        case SignalColor::Yellow:
                color = initial_color;
                break;
        default:
                color = SignalColor::Red; // Red by default, just in case
        }
}
// Ensures the traffic light's signal sequence
void Trafficlight::change() {
        // Red --> green, green --> yellow, yellow --> red
        if (color == SignalColor::Red)
                color = SignalColor::Green;
        else if (color == SignalColor::Green)
                color = SignalColor::Yellow;
        else if (color == SignalColor::Yellow)
                color = SignalColor::Red;
}
// Returns the light's current color so a client can
// act accordingly
SignalColor Trafficlight::get_color() const {
        return color;
}
```

< 15.9 >

```
#include <iostream>
#include "trafficlight.h"
void print(Trafficlight lt) {
	SignalColor color = lt.get_color();
	std::cout << "+-----+₩n";
	std::cout << "| |₩n";
	if (color == SignalColor::Red)
		std::cout << "| (R) |₩n";
	else
		std::cout << "| ( ) |₩n";
	std::cout << "| |₩n";
	if (color == SignalColor::Yellow)
		std::cout << "| (Y) |₩n";
	else
		std::cout << "| ( ) |₩n";
	std::cout << "| |₩n";
	if (color == SignalColor::Green)
		std::cout << "| (G) |₩n";
	else
		std::cout << "| ( ) |₩n";
	std::cout << "| |₩n";
	std::cout << "+-----+₩n";
}
int main() {
	Trafficlight light(SignalColor::Green);
	while (true) {
		print(light);
		light.change();
		std::cin.get();
	}
}
```

< 실행결과 >

```
+-----+

| |

| ( ) |

| |

| ( ) |

| |

| (G) |

| |

+-----+
```

+-----+

| |

| ( ) |

| |

| (Y) |

| |

| ( ) |

| |

+-----+

+-----+

| |

| (R) |

| |

| ( ) |

| |

| ( ) |

| |

+-----+

(반복)

< 설명 >

우선 헤더파일 하나와 소스파일 두개가 나뉘어져있다. 헤더파일먼저 살펴보면 선언들만 해주었다. 열거형 클래스로 3개의 색을 선언해주고 Trafficlight class를 선언해주었다. private형으로 앞서 선언한 열거형 클래스를 넣어주고 public형으로 생성자와 change 함수, get_color 함수를 선언해주었다. 아마 소스파일에서 이 함수들에 대한 정의를 해줄 것으로 보인다. trafficlight.cpp파일을 보면 전처리문을 사용하여 헤더파일을 불러왔다. 그래서 class로 선언된 Trafficlight의 생성자를 정의해준다. 파라미터로 SignalColor 클래스 객체를 받아서 그것이 Red, Green, Yellow인 경우에 각 받

은 색이 color라는 변수에 들어가게 하고 이때 이 color 변수는 헤더파일에서 이미 private에 선언이 되었다. 셋 중에 포함이 안되는 경우 기본값으로 빨간불이 들어간다.

이제 change 함수를 정의해준다. color 변수가 RED인 경우 Green으로 바꿔주고 Green인 경우 Yellow로, Yellow인 경우 Red로 바꿔준다. 아무 파라미터 없이 동작한다. get_color()함수도 아무 파라미터를 받지 않고 현재 color변수에 할당된 값을 SignalColor class 형으로 return 해준다.

trafficmain 소스파일을 살펴보자. Trafficlight class 객체를 파라미터로 받아서 출력하는 print 함수를 정의하고 있다. get_color()함수를 이용하여 어떤 색이 할당되어있는지 가져온 뒤, 그 색이 Red, Yellow, Green인 각각의 경우에 따라서 신호등처럼 출력을 해준다. 그래서 main 함수 안을 보면 Trafficlight class의 객체 light를 초기값 파라미터로 Green으로 선언해준다. 이후 while문을 통해 방금 만든 print함수를 통해 신호등 그림을 출력하고, change함수를 통해 자동으로 그 다음 색으로 바꿔둔다. 그리고 cin.get()을 통해 명령이 받아진 만큼 while문이 돌아간다. 그래서 Enter만 칠 경우 Yellow로 바뀐 모습이 보이는데, 5글자를 치고 Enter를 치면 6개의 신호등이 바뀌어가며 나온다.

A-2. Listing 16.5

< 코드 >

```
#include <iostream>
#include <vector>
/*
* Comparer objects manage the comparisons and element
* interchanges on the selection sort function below.
*/
class Comparer {
        // Keeps track of the number of comparisons
        // performed
        int compare_count;
        // Keeps track of the number of swaps performed
        int swap_count;
        // Function pointer directed to the function to
        // perform the comparison
        bool (*comp)(int, int);
public:
        // The client must initialize a Comparer object with a
        // suitable comparison function.
        Comparer(bool (*f)(int, int)) :
                compare_count(0), swap_count(0), comp(f) {}
        // Resets the counters to make ready for a new sort
        void reset() {
                compare_count = swap_count = 0;
        }
        // Method that performs the comparison. It delegates
        // the actual work to the function pointed to by comp.
        // This method logs each invocation.
```

```
        bool compare(int m, int n) {
                compare_count++;
                return comp(m, n);
        }
        // Method that performs the swap.
        // Interchange the values of
        // its parameters a and b which are
        // passed by reference.
        // This method logs each invocation.
        void swap(int& m, int& n) {
                swap_count++;
                int temp = m;
                m = n;
                n = temp;
        }
        // Returns the number of comparisons this object has
        // performed since it was created.
        int comparisons() const {
                return compare_count;
        }
        // Returns the number of swaps this object has
        // performed since it was created.
        int swaps() const {
                return swap_count;
        }
};
/*
* selection_sort(a, compare)
* Arranges the elements of vector a in an order determined
* by the compare object.
* a is a vector of ints.
* compare is a function that compares the ordering of
* two integers.
* The contents of a are physically rearranged.
*/
void selection_sort(std::vector<int>& a, Comparer& compare) {
        int n = a.size();
        for (int i = 0; i < n - 1; i++) {
                // Note: i,small, and j represent positions within a
                // a[i], a[small], and a[j] represents the elements at
                // those positions.
                // small is the position of the smallest value we've seen
                // so far; we use it to find the smallest value less
                // than a[i]
                int small = i;
                // See if a smaller value can be found later in the array
                for (int j = i + 1; j < n; j++)
                        if (compare.compare(a[j], a[small]))
                                small = j; // Found a smaller value
                                // Swap a[i] and a[small], if a smaller value was found
                if (i != small)
                        compare.swap(a[i], a[small]);
        }
}
/*
```

```
* print
* Prints the contents of an integer vector
* a is the vector to print.
* a is not modified.
*/
void print(const std::vector<int>& a) {
        int n = a.size();
        std::cout << '{';
        if (n > 0) {
                std::cout << a[0]; // Print the first element
                for (int i = 1; i < n; i++)
                        std::cout << ',' << a[i]; // Print the rest
        }
        std::cout << '}';
}
/*
* less_than(a, b)
* Returns true if a < b; otherwise, returns
* false.
*/
bool less_than(int a, int b) {
        return a < b;
}
/*
* greater_than(a, b)
* Returns true if a > b; otherwise, returns
* false.
*/
bool greater_than(int a, int b) {
        return a > b;
}
int main() {
        // Make a vector of integers from an array
        std::vector<int> original{ 23, -3, 4, 215, 0, -3, 2, 23, 100, 88, -10 };
        // Make a working copy of the original vector
        std::vector<int> working = original;
        std::cout << "Before: ";
        print(working);
        std::cout << '₩n';
        Comparer lt(less_than), gt(greater_than);
        selection_sort(working, lt);
        std::cout << "Ascending: ";
        print(working);
        std::cout << " (" << lt.comparisons() << " comparisons, "
                << lt.swaps() << " swaps)₩n";
        std::cout << "----------------------------₩n";
        // Make another copy of the original vector
        working = original;
        std::cout << "Before: ";
        print(working);
        std::cout << '₩n';
        selection_sort(working, gt);
        std::cout << "Descending: ";
        print(working);
        std::cout << " (" << gt.comparisons() << " comparisons, "
```

```
                << gt.swaps() << " swaps)₩n";
        std::cout << "----------------------------₩n";
        // Sort a sorted vector
        std::cout << "Before: ";
        print(working);
        std::cout << '₩n';
        // Reset the greater than comparer so we start counting at
        // zero
        gt.reset();
        selection_sort(working, gt);
        std::cout << "Descending: ";
        print(working);
        std::cout << " (" << gt.comparisons() << " comparisons, "
                << gt.swaps() << " swaps)₩n";
}
```

< 실행결과 >

Before: {23,-3,4,215,0,-3,2,23,100,88,-10}

Ascending: {-10,-3,-3,0,2,4,23,23,88,100,215} (55 comparisons, 7 swaps)

Before: {23,-3,4,215,0,-3,2,23,100,88,-10}

Descending: {215,100,88,23,23,4,2,0,-3,-3,-10} (55 comparisons, 5 swaps)

Before: {215,100,88,23,23,4,2,0,-3,-3,-10}

Descending: {215,100,88,23,23,4,2,0,-3,-3,-10} (55 comparisons, 0 swaps)

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 94844개)이(가

이 창을 닫으려면 아무 키나 누르세요...

< 설명 >

위에서부터 차례로 설명. Comparer class를 만듦. private로 int형 변수 compare_count, swap_count로 카운트하는 변수, boolean으로 return하는 함수 포인터 comp 선언. public으로는 생성자를 정의해주고 reset 함수 정의, 이것은 카운트를 모두 0으로 해준다. compare 함수는 m과 n이 comp 함수에 의해 비교된 결과를 return하며 compare_count를 1 올린다. swap 함수는 swap_count를 1 올리면서 파라미터로 받은 두 int형 변수에 할당된 값을 바꾼다. comparisons 함수는 compare_count를 return하며 const형이다. swaps 함수도 마찬가지이다. class 밖에서 selection_sort라는 함수가 정의되어있다. 파라미터로 int형 벡터와 Comparer class로 지정된 객체를 받는다. 벡

터의 사이즈동안 for문을 돌리는 것이다. 가장 작은 small값을 가장 첫번째 벡터요소로 두고 시작하여서 뒤로 가면서 가장 작은 small 값을 찾아서 그 위치에서의 요소의 값을 서로 바꾼다. print 함수는 int형 벡터를 parameter로 받아서 쭉 출력해주는 계속 봐온 함수이다. less_than 함수는 parameter로 받은 두 정수에서 b가 클 경우 true, greater_than 함수는 a가 클 경우 true를 return 한다. 메인 함수를 살펴보자.

우선 int형 벡터 original을 사이즈 11로 다양한 정수 요소를 넣어두었다. 그리고 working 벡터에 그대로 복사해준다. print 함수를 이용하여 working 벡터를 출력해준다. less_than함수와 greater_than 함수를 parameter로 받는 Comparer class 객체 lt와 gt를 선언해준다. 이후 selection_sort 함수를 통해 working 벡터에 있는 요소를 lt함수 기준으로 정렬한다. lt함수는 오름차순으로 정렬하기 위한 기준이다. 따라서, 이후 working을 출력하면 오름차순으로 정렬이 된 것이 파악이 가능하고, 객체 lt의 comparisons와 swaps method를 출력해주었을 때 비교횟수랑 바꾼 횟수가 출력되는 것을 알 수 있다. (각각의 count를 출력한것임)

이후 다시 working 벡터에 원본을 넣어주고 이번엔 객체 gt를 기준으로 정렬한다. gt는 내림차순을 기준으로 하도록 하였다. 따라서, selection_sort 함수 실행 이후 working벡터가 내림차순으로 정렬되어 출력되고 있고, gt의 comparsions, swaps method도 잘 사용이 되어 비교 횟수, 스왑 횟수가 잘 나타난다. 이 상태에서 working 벡터를 초기화하지 않고 gt의 method인 reset을 통해 count들만 초기화 한다음에 다시 selection_sort 함수에 넣고 실행시키면 이미 정렬이 되어있기 때문에 비교횟수는 똑같으나 swap 횟수는 0인 것을 확인할 수 있다.

A-3. Listing 16.9, 16.10

< 16.9 >

```
#ifndef UNIFORM_RANDOM_DEFINED_
#define UNIFORM_RANDOM_DEFINED_
#include <random>
class UniformRandomGenerator {
        // A uniform distribution object
        std::uniform_int_distribution<int> dist;
        // A Mersenne Twister random number generator with a seed
        // obtained from a random_device object
        std::mt19937 mt;
public:
        // The smallest pseudorandom number this generator can produce
        const int MIN;
        // The largest pseudorandom number this generator can produce
        const int MAX;
        // Create a pseudorandom number generator that produces values in
        // the range low...high
        UniformRandomGenerator(int low, int high) : dist(low, high),
                mt(std::random_device()()),
                MIN(low), MAX(high) {}
```

```
        // Return a pseudorandom number in the range MIN...MAX
        int operator()() {
                return dist(mt);
        }
};
#endif
```

< 16.10 >

```
#include <iostream>
#include <iomanip>
#include "uniformrandom.h"
int main() {
        // Pseudorandom number generator with range 0...9,999
        UniformRandomGenerator rand(0, 9999);
        // Total counts over all the runs.
        // Make these double-precision floating-point numbers
        // so the average computation at the end will use floating-point
        // arithmetic.
        double total5 = 0.0, total9995 = 0.0;
        // Accumulate the results of 10 trials, with each trial
        // generating 1,000,000,000 pseudorandom numbers
        const int NUMBER_OF_TRIALS = 10;
        for (int trial = 1; trial <= NUMBER_OF_TRIALS; trial++) {
                // Initialize counts for this run of a billion trials
                int count5 = 0, count9995 = 0;
                // Generate one billion pseudorandom numbers in the range
                // 0...9,999 and count the number of times 5 and 9,995 appear
                for (int i = 0; i < 1000000000; i++) {
                        // Generate a pseudorandom number in the range 0...9,999
                        int r = rand();
                        if (r == 5)
                                count5++; // Number 5 generated, so count it
                        else if (r == 9995)
                                count9995++; // Number 9,995 generated, so count it
                }
                // Display the number of times the program generated 5 and 9,995
                std::cout << "Trial #" << std::setw(2) << trial << " 5: "
                        << std::setw(6) << count5
                        << " 9995: " << std::setw(6) << count9995 << '₩n';
                total5 += count5; // Accumulate the counts to
                total9995 += count9995; // average them at the end
        }
        std::cout << "-------------------₩n";
        std::cout << "Averages for " << NUMBER_OF_TRIALS << " trials: 5: "
                << std::setw(6) << total5 / NUMBER_OF_TRIALS << " 9995: "
                << std::setw(6) << total9995 / NUMBER_OF_TRIALS << '₩n';
}
```

< 실행결과 >

Trial # 1 5: 99001 9995: 100094

Trial # 2 5: 100095 9995: 100317

Trial # 3 5: 100319 9995: 100259

Trial # 4 5: 100299 9995: 99744

Trial # 5 5: 99407 9995: 100146

Trial # 6 5: 100362 9995: 100146

Trial # 7 5: 99740 9995: 99535

Trial # 8 5: 100342 9995: 99941

Trial # 9 5: 100064 9995: 99756

Trial #10 5: 99571 9995: 99918

Averages for 10 trials: 5: 99920 9995: 99985.6

C:₩Users₩Administrator₩source₩repos₩Ex002₩Release₩Ex002.exe(프로세스 104680개)이

(가) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...

- 속도가 너무 느려서 속도 release 모두에서 속도 우선 최적화 하였습니다.

< 설명 >

헤더 파일에서는 UNIFORM_RANDOM_DEFINED_라는 이름이 전처리에 의해 define 되어있는지 확인을 하고 되어있지 않다면 헤더파일의 전체 내용을 불러올 수 있도록 한다. UniformRandomGenerator class를 선언해주고 private에 standard class인 uniform_int_distribution 객체와 mt19937 객체가 선언된다. rand의 문제점을 보완한 난수 생성 엔진이라고 한다. public에 const int형으로 MIN과 MAX를 선언해주고 생성자를 선언해준다. 기본적으로 mt의 파라미터로는 또 standard class인 random_device(아무것도 파라미터로 받지 않음) MIN은 첫번째 파라미터로 받는 low, MAX는 두번째 파라미터로 받는 high를 넣어준다. 그리고 dist는 low와 high를 파라미터로 받는 것이 기본이다. method로 아무 파라미터도 안받는 operator()()함수는 dist(mt)값을 return한다. 아무래도 mt를 dist class에 넣으면 int형으로 retrun이 되는 모양이다.

이제 main 함수를 살펴본다. 앞서 헤더파일에서 선언한 class의 객체로 이름이 rand이고 MIN이 0, MAX가 9999인 객체를 선언해준다. 이후 double형 변수 total5와 total9995를 초기화해준다. 아마 몇번 나왔는지의 평균값을 넣을 모양이다. const int형으로 NUMBER_OF_TRIALS도 초기화해준다.

이것은 몇회 실행할지 적는 것이다. for문을 통해 앞서 선언한 범위 내에서 랜덤 난수를 돌린다. for문을 통해 1000000000번 돌리는데 이 과정에서 rand를 이용해 5가 나오는 경우 count5를 1 올려주고 9995가 나오는 경우 count9995를 1 올려준다. 이 과정을 10번 더 반복하여 나온 횟수의 평균값을 계산해줄 것이다. 출력하는 부분은 기본적인 부분이기에 설명을 간단히 하겠다. 그래서 몇 번째 trial인지 알려주면서 그 trial 때 5와 9995가 각각 몇 번 나왔는지 출력해주고, total에 더해서 나중에 for문 밖으로 나와서 평균값을 제공해준다.

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

(pp. 470-474), Exercises 1-16
(write output analysis for all exercises)

1. Suppose Widget is a class of objects, and function proc accepts a single Widget object as a parameter. Without knowing anything about class Widget, which of the following definitions of function proc is considered better, and why? Option #1: void proc(Widget obj) { /* Details omitted . . . */ } Option #2: void proc(const Widget& obj) { /* Details omitted . . . */ }

< 설명 >

Option #2가 낫다. Widget class에 대한 아무런 정보가 없으므로 우리가 proc 함수를 통해 어떤 동작을 취하고자 하는지 역시 모른다. proc이 obj의 내용을 변경하는 함수는 아니므로 (왜냐하면 두 옵션 중 obj를 직접 수정할 수 있는 경우가 없음) 그렇다. 그래도 obj를 레퍼런스로 불러와서 const로 취해주는 것이 실수로 Widget이 수정되는 경우도 막을 수 있고 레퍼런스를 취했기 때문에 메모리 낭비를 막을 수 있다. (새로 객체를 만들지 않기 때문)

2. Suppose you have the following definition for class Assembly: class Assembly { public: int value; }; and the variable ptr_a declared and initialized as so: Assembly *ptr_a = new Assembly;

: Assembly class형이고 Assembly의 생성을 가리키는 포인터 ptr_a가 있다.

(a) What statement using the dot operator (.) could you use to assign 5 to the value field of the object to which ptr_a points?

: (*ptr_a).value = 5; (괄호 반드시 넣어줘야함)

(b) What statement using the arrow operator (->) could you use to assign 5 to the value field of the object to which ptr_a points?

: ptr_a->value = 5; (아무래도 a보다 b가 더 간편하긴 하다)

3. Suppose you have the following definition for class Gadget: class Gadget { int value; public: Gadget(int v): value(v) {} int get() const { return value; } }; Consider the following code fragment: Gadget x(5); int y = x.get();

(a) What value does the second statement assign to y?

: y에 5를 할당한다.

(b) Rewrite Gadget::get so it explicitly uses the this pointer.

:
```
int get() const {
		return this->value;
	}
```

(c) During the execution of Gadget::get on the second line, what is the value of the method's this pointer?

: 5

(d) What can a client do to alter the value field of a Gadget object after the object's creation?

: 없는 것 같습니다. get 함수도 const 라 value값을 레퍼런스로 받으려면 변수도 const로 선언되어야해서 값을 바꿀수가 없는 것 같습니다. 값을 바꾸는 함수도 없습니다.

4. What are the consequences of declaring a method to be const?

: const 객체가 const가 표시된 method를 호출할 경우 사용할 수 있게 해준다. 만약 method에 const가 없다면 const 객체가 그 method를 사용할 수 없다.

5. Why are const methods necessary in C++?

: 앞선 3번에서 본 것처럼 그 객체가 가지고 있는 값을 불러올 수는 있으나 변경하지 못하게 할 때 사용하기 유용하다. 말그대로 method로 인해 객체의 값이 변경되지 않도록 하기 위해서이다.

6. Given the following Counter class declaration:

```
class Counter
{
	int value;
```

```
public:
        Counter() : value(0) {}
        void increment()
        {
                value++;
        }
        int get() const
        {
                return value;
        }
};
```

and Counter objects declared as shown here:

```
Counter c1;
const Counter c2;
```

determine if each of the following statements is legal.

(a) `c1.increment();`

`c2.increment();`

: c1.increment()는 가능하지만 c2.increment()는 불가능하다. 왜냐하면 const를 이용해서 c2를 선언했기 때문에 c2에 있는 값을 변경하는 것이 불가능하기 때문이다. 또한 increment 함수에 const로 지정되어있지 않음

(b) `int x = c1.get();`

`int y = c2.get();`

: 둘 다 가능하다. get() 함수가 const로 지정되어있기 때문에 const로 선언된 c2의 경우에도 사용 가능하고, 그렇지 않은 c1의 경우에도 사용이 가능하다.

7. Given the following Counter class declaration:

```
class Counter {
        int value;
public:
        Counter() : value(0) {}
```

```
        void increment() {
                value++;
        }
        int get() {
                return value;
        }
};
```

and Counter objects declared as shown here:

```
Counter c1;
const Counter c2;
(a) c1.increment();
    c2.increment();
```

: c1.increment()는 가능하지만 c2.increment()는 불가능하다. 왜냐하면 const를 이용해서 c2를 선언했기 때문에 c2에 있는 값을 변경하는 것이 불가능하기 때문이다. 또한 increment 함수에 const로 지정되어있지 않음

```
(b) int x = c1.get();
    int y = c2.get();
```

: 앞선 문제와 달리 c2.get()은 불가능하다. 왜냐하면 get함수가 정의될 때 const로 지정이 되어있지 않기 때문에 const로 지정된 c2에게 그 함수를 적용하는 것은 불가능하다. 다만 c1.get()은 가능하다.

8. Consider the following class declaration:

```
class Counter
{
        int value;
public:
        Counter() : value(0) {}
        void increment()
        {
                value++;
        }
```

```
        int decrement()
        {
                value--;
        }
        int get() const
        {
                return value;
        }
};
```

(a) Show how you would properly separate the code of the Counter class into two source files: its declaration in a counter.h file and its method implementations in a counter.cpp file.

< counter.h >

```
class Counter
{
        int value;
public:
        Counter() : value(0) {}
        void increment();
        int decrement();
        int get() const;
};
```

< counter.cpp >

```
#include "counter.h"

void Counter::increment()
{
        value++;
}
int Counter::decrement()
{
        value--;
```

```
}
int Counter::get() const
{
        return value;
}
```

(b) Would client code ordinarily #include both the counter.h and counter.cpp files in its source code?

: 헤더 파일에는 #include가 필요하지 않고 cpp 파일에만 필요하다.

(c) How can you protect your counter.h file to prevent clients from accidentally #include-ing the contents of the header file more than once?

: 교수님께서 설명하신 #ifndef #define #endif 를 이용해주면 된다. 선언되지 않은 경우에만 define 할 수 있도록 한다. 따라서 헤더파일의 중복을 막는다.

(d) What advantages does separating the class declaration and method implementations provide?

: 실행시간이 줄어들 수도 있으며 메모리 할당도 유리하다.

9. What are the consequences of declaring a method to be static?

: class로 생성되는 모든 객체가 공유할 수 있다. 마치 전역변수, 함수와 같다.

10. Consider the following class declaration:

```
class ValType
{
public:
        int value1;
        static int value2;
};
```

(a) If a client creates 100 ValType objects, how many value1 fields will exist in memory?

: 100개의 메모리 공간을 사용할 것이다. 각각의 value1 값이 존재

(b) If a client creates 100 ValType objects, how many value2 fields will exist in memory?

: 1개의 메모리 공간을 사용한다. 따라서, 100개 중 1개의 객체의 value2 값을 바꾼다면 모든 객체의 value2 값이 바뀔 것이다.

11. How is a C++ struct similar to a class?

: 작동 방식과 선언 방식이 유사하다.

12. How does a C++ struct differ from a class?

: class의 디폴트값은 private이지만 struct의 디폴트값은 public이다.

13. Given the following custom type declarations:

```
class X
{
        int value1;
public:
        static int value2;
        int value3;
        X() : value1(5) {}
        void f()
        {
                value1 = 0;
        }
        static void g()
        {
                value2 = 0;
        }
};
int X::value2 = 3;
struct Y
{
        int quantity1;
        Y() : quantity1(5) {}
        void f()
        {
                quantity1 = 0;
        }
private:
```

```
        int quantity2;
};
```

and the following variable declarations:

```
X x_obj;
Y y_obj;
```

determine if each of the following statements is legal.

(a) x_obj.value1 = 0; illegal (private 이므로)

(b) x_obj.value2 = 0; legal (public 이므로)

(c) x_obj.value3 = 0; legal (public 이므로)

(d) X::value1 = 0; illegal (private 이므로)

(e) X::value2 = 0; legal (static int기 때문에 가능, 그리고 따로 초기화도 됨)

(f) X::value3 = 0; illegal (변수가 static이 아니라 선언되지 않은 변수임)

(g) y_obj.quantity1 = 0; legal (struct는 디폴트가 public)

(h) y_obj.quantity2 = 0; illegal (private 이므로)

(i) Y::quantity1 = 0; illegal (변수가 static이 아니라 선언되지 않은 변수임)

(j) Y::quantity2 = 0; illegal (변수가 static이 아니라 선언되지 않은 변수임+private)

(k) x_obj.f(); legal (public에 있는 함수기 때문에 가능)

(l) x_obj.g(); legal (상동)

(m) X::f(); illegal (함수가 static이 아니라 선언되지 않은 함수임)

(n) X::g(); legal (함수가 static이기 때문에 가능)

14. What privileges does function f have with respect to the class that declares f to be its friend?

: private로 선언된 공간까지 접근하여 사용할 수 있다.

15. Consider the following class declaration:

```
class ValType
{
        int value;
public:
        ValType() : value(0) {}
        void set(int v1)
```

```
        {
                value = v1;
        }
        void show() const
        {
                std::cout << value << '₩n';
        }
        friend int f(const ValType& x);
};
```

and determine the legality of each of the following function definitions:

(a)
```
int f(const ValType& x)
{
        return 2 * x.value;
}
```

: legal. 가능하다. class 내에서 이 함수에 대해 friend로 해주었음.

(b)
```
int g(const ValType& x)
{
        return 2 * x.value;
}
```

: illegal. 불가능하다. class 내의 private 변수에 접근하려고 함

(c)
```
int f(const ValType& x)
{
        x.show();
        return 0;
}
```

: legal. 가능하다. class 내에서 이 함수에 대해 friend로 해주었음.

(d)
```
int g(const ValType& x)
{
        x.show();
        return 0;
}
```

: legal. 가능하다. class 내에서 public인 공간에만 접근하고 있음

```
(e) int f(const ValType& x, int n)
{
        return n * x.value;
}
```

: illegal. 불가능하다. class 내의 private 변수에 접근하려고 함. 오버로딩도 같은 parameter가 friend로 선언된 경우만 가능하다!

16. What are the risks associated with using the friend construct?

: private 공간에 있는 member를 사용할 때 수정되는 것을 주의해야한다.

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

(pp. 493-494), Exercises 1
(write output analysis for all exercises)

1. Create a large unsigned integer type named BigUnsigned. BigUnsigned objects represent unsigned integers with arbitrary precision; that is, unlike the standard C++ unsigned integer primitive types, a BigUnsigned object can represent an unsigned integer as large as necessary. Unlike the floating-point types, a BigUnsigned value retains all its digits of precision.

Internally, the BigUnsigned class should hold a std::vector of integers. Each integer in the vector is one of 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. Each element in the vector represents a digit in a place value within the integer; for example, if the internal vector contains the following elements in the following order:

4,9,1,1,4,3,0,5

we would interpret the associated BigUnsigned object as the mathematical nonnegative integer 49,114,305. Your BigUnsigned class implementation should provide the following features:

• The class should provide a constructor that accepts no arguments that initializes the BigUnsigned object's vector to contain a single element equal to zero.

• The class should provide a constructor that accepts a single unsigned integer. This constructor should populate its internal vector with the appropriate elements to correspond to the value of its parameter.

• The class should provide a constructor that accepts a BigUnsigned argument. Clients use this constructor to create a new BigUnsigned object from an exiting BigUnsigned object.

• The class should provide a constructor that accepts a std::string object representing an integer. Clients use this constructor when they need to create a large integer whose value exceeds the range of unsigned long long. The string argument should contain only digits.

• The class should provide access to a friend function named operator+ that adds two BigUnsigned objects and returns the BigUnsigned result.

• The class should provide access to a friend function named operator<< that allows clients to print a BigUnsigned value as easily as a built-in integer type.

< 전체 코드 >

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

class BigUnsigned {
	std::vector<int> v1;
public:
	BigUnsigned() : v1({ 0 }) {}
	BigUnsigned(unsigned int n) : v1(v1) {
		int x = 0;
		while (n > 9) {
			x = n % 10;
			n /= 10;
			v1.push_back(x);
		}
		x = n % 10;
		v1.push_back(x);
		std::reverse(begin(v1), end(v1));
	}
	BigUnsigned(std::string str) : v1(v1) {
		int size = str.size();
		for (int i = 0; i < size; i++) {
			int n = stoi(str.substr(i, 1));
			v1.push_back(n);
		}
		/*int num = stoi(str), x = 0;
		while (num > 9) {
			x = num % 10;
			num /= 10;
			v1.push_back(x);
		}
		x = num % 10;
		v1.push_back(x);
```

```
                std::reverse(begin(v1), end(v1));*/
        }
        BigUnsigned(const BigUnsigned& bu) : v1(v1) {
                v1 = bu.v1;
        }
        friend BigUnsigned operator+(const BigUnsigned& bu1, const BigUnsigned& bu2);
        friend std::ostream& operator<<(std::ostream& os, const BigUnsigned& bu);
};
BigUnsigned operator+(const BigUnsigned& bu1, const BigUnsigned& bu2) {
        BigUnsigned result;
        result.v1.pop_back();
        int c = 0, a = 0, aa = 0;
        bool n = (bu1.v1.size() >= bu2.v1.size());
        if (n) {
                a = bu2.v1.size();
                aa = bu1.v1.size();
                for (int i = 0; i < a; i++) {
                        int b = bu1.v1[aa - i - 1] + bu2.v1[a - i - 1] + c;
                        c = 0;
                        if (b > 9) {
                                c = 1;
                                b %= 10;
                        }
                        result.v1.push_back(b);
                }
                int aaa = aa - a;
                for (int i = aaa - 1; i >= 0; i--) {
                        int b = bu1.v1[i] + c;
                        c = 0;
                        if (b > 9) {
                                c = 1;
                                b %= 10;
                        }
                        result.v1.push_back(b);
                }
                if (c == 1) result.v1.push_back(1);
        }
        else {
                a = bu1.v1.size();
                aa = bu2.v1.size();
                for (int i = 0; i < a; i++) {
                        int b = bu1.v1[a - i - 1] + bu2.v1[aa - i - 1] + c;
                        c = 0;
                        if (b > 9) {
                                c = 1;
                                b %= 10;
                        }
                        result.v1.push_back(b);
                }
                int aaa = aa - a;
                for (int i = aaa - 1; i >= 0; i--) {
                        int b = bu2.v1[i] + c;
                        c = 0;
                        if (b > 9) {
                                c = 1;
```

```
                                b %= 10;
                        }
                        result.v1.push_back(b);
                }
                if (c == 1) result.v1.push_back(1);
        }
        /*for (int i = aaa - 1; i >= 0; i--) {
                if (n)
                        result.v1.push_back(bu1.v1[i]);
                else
                        result.v1.push_back(bu2.v1[i]);
        }*/
        std::reverse(begin(result.v1), end(result.v1));
        return result;
}
std::ostream& operator<<(std::ostream& os, const BigUnsigned& bu) {
        int size = bu.v1.size();
        for (int i = 0; i < size; i++) {
                os << bu.v1[i];
        }
        return os;
}
int main() {
        std::string str2 = "12345678901234567890123456789012345678901234567890";
        BigUnsigned a(str2);
        BigUnsigned b;
        BigUnsigned c(a);
        BigUnsigned d(1234567890);
        BigUnsigned e(110);
        std::string str1 = "99999999999999999999999999999999999999999999999999999999999999";
        BigUnsigned f(str1);
        BigUnsigned g(1);
        std::cout << a << std::endl;
        std::cout << b << std::endl;
        std::cout << c << std::endl;
        std::cout << d << std::endl;
        std::cout << e + d << std::endl;
        std::cout << f + g << std::endl;
}
```

- 테스트 코드 만든 것까지 전부 넣었습니다.

< 설명 >

오랜 시간 고민하여 class를 직접 만들어보았다. 제공해야할 특성 전부 고려하였다. 문제에서 주어진 특성에 설명이 다 되어있어 간략하게 하겠다.

우선 BigUnsigned class를 정의해주었다. private에 사용할 벡터 하나를 선언해준다. 이후 public에 문제에서 요구한 생성자 4개를 구현했다.

1. parameter가 없을 때 자동으로 벡터에 0만 있는 것을 구현하였고,

2. unsigned int가 들어왔을 때 한 자리씩 벡터의 요소에 차곡차곡 넣고,

3. string형 문자열을 받았을 때, 그 string에서 substr method를 이용하여 한 자리씩 가져와서 int 형으로 변경한 뒤에 vector에 하나씩 넣는 작업을 한다.

4. 같은 class인 BigUnsigned형을 받았을 때도 안에 들어있는 벡터를 복사해주면서 생성자를 구현하였다.

이후 문제에서 요구한대로 operator 오버로딩을 다른 class에서도 사용할 수 있도록 friend 선언해준 뒤 아래에 class 밖에서 구현하였다.

+ operator 부터 살펴보면 두 개의 BigUnsigned형을 + 연산자로 더할 것인데 비어있는 BigUnsigned를 선언하기 위해 파라미터 없이 선언하고 안에 들어있는 vector의 0을 지운다. 이후 더하기를 해 줄 두 BigUnsigned 내에 있는 벡터의 크기를 비교하여 작은 벡터의 크기만큼 뒤에서부터 요소끼리 더해준다. 이 때, 9+9 처럼 자릿수가 올라가는 경우가 있을 수 있으므로 이것의 경우 앞자리수에 1을 더해줘야하므로 변수 c에 따로 저장을 해두고 앞 요소에서의 연산에 포함될 수 있도록 한다. 따라서, 뒤에서부터 자릿수를 올려가며 한 요소씩 연산을 하고 사이즈가 큰 벡터에 남아있는 요소들을 그대로 붙여준다. 이 때도 자릿수 올림이 있을 수 있다는 것을 생각하고 진행하였다. 편의상 뒤에서부터 진행하므로 더해서 새로 만들어지는 벡터는 뒷자리수부터 정렬이 되어있으므로 마지막에 algorithm을 이용하여 reverse 해준다. 그리고 return

사실 자릿수 올림이 가장 어려웠던 부분인데 코드가 비효율적이어보이긴 해도 기능 구현을 했다는 점에서 매우 만족한다. 어떤 수를 넣어도 정상적으로 자릿수 올림이 가능하다!

이후 << 연산자 오버로딩은 강의자료에 있던대로 참고하여 BigUnsigned 형을 받았을 때 그 벡터의 요소가 앞에서부터 하나씩 << 처리 되도록 하여 정상적으로 정수처럼 보이게 나올 수 있도록 하였다! 정수를 int형에 저장하지 않으므로 문제의 의도에 맞게 범위에 벗어나는 수도 저장할 수 있다!

따라서 테스트 코드는 main 함수에 작성하였는데, a로 string형을 받는 것을 확인, b로 디폴트 생성자 확인, c로 같은 class를 받을 때도 정상적으로 복사되는 것을 확인, d로 int형도 정상적으로 받는 다는 것을 확인하였다. 이후 d와 e, f와 g를 + 연산자 하면서 자릿수 올림이 잘 되는지, 엄청 큰 수가 저장되는지 잘 확인을 하였고 << 도 사용하는 것처럼 정상적으로 잘 작동하는 것을 알 수 있다.

< 실행결과 >

12345678901234567890123456789012345678901234567890

0

12345678901234567890123456789012345678901234567890

1234567890

1234568000

10000000000000000000000000000000000000000000000000000000000000000

C:₩Users₩Administrator₩source₩repos₩Ex002₩Debug₩Ex002.exe(프로세스 68748개)이(가

) 종료되었습니다(코드: 0개).

이 창을 닫으려면 아무 키나 누르세요...