객체지향프로그래밍 숙제 #4

강대기

2007년 11월 1일

제 1 절 필독: 숙제 제출에 대해 반드시 알아야 할 사항

앞의 과제에서 이미 읽었다고 해도, 다시 한 번 읽기 바랍니다.

본 숙제는 E11, E12, E13 세 반 모두 일괄적으로 11월 9일까지 제출해야 하도록 한다.

숙제를 늦게 제출할 경우, 하루에 20% 씩 점수가 깍인다. 이를테면, 100점 만점에서 하루 늦는 경우는 만점을 맞아도 80 점이 된다. 이틀 늦는 경우는 만점을 맞아도 60점이다. 주말의 경우는 토요일과 일요일을 합해서 하루로 친다.

숙제를 제출할 때는 연습 문제를 푼 내용들을 하나의 텍스트 파일로 작성하여 이메일 **dkkang@dongseo.ac.kr**로 보낸다. 프로그램 숙제들은 실행 파일은 보내지 말고, 소스 코드만 작성하여 전체를 하나의 ZIP 파일로 묶어서 보낸다. RAR나 알집(ALZ) 형식으로 묶지 말고, 되도록이면 ZIP 파일로 묶어서 보낸다.

따라서, 보낼 파일의 개수는 연습문제를 풀은 텍스트 파일 하나와 프로그램 소스들을 묶은 ZIP 파일 하나로 두 개가 된다.

소스 코드의 파일 이름은, 별도로 지정하지 않는 한, 프로그래밍 문제의 번호를 토대로 만든다. 예를 들어 프로그래밍 문제 7.4의 경우, `7_4.cpp` 라고 이름을 지으면 된다.

한글 인코딩은 되도록이면 EUC-KR을 코딩을 따르되, UTF-8도 허용가능하다. 이메일로 제출할 때, 제목을 "[OOP2007]"로 시작하고 "HW4", 반, 학번, 이름 등을 순서대로 기재한다. 각 필드는 '.'로 끊어준다.

예를 들어 E11반 학번 20061234인 홍길동이 숙제 #4을 제출하는 경우 다음과 같이 메일 제목을 쓰면 된다.

```
제목: [OOP2007]HW4.E11.20061234.홍 길 동
```

제 2 절 숙제

2.1 읽기 숙제 (제출할 필요 없음)

"C++기초플러스 - 5판"의 7장과 8장

2.2 풀어보기 숙제 (제출할 필요 없음)

"C++기초플러스 - 5판"의 7장과 8장의 모든 연습 문제들

2.3 연습 문제 숙제 (텍스트 파일로 제출)

다음의 문제들을 풀어서 텍스트 파일로 제출한다.

2.3.1 포인터 및 함수 선언

다음의 선언이나 정의, 또는 명령문들이 의미하는 바를 설명하라.

1. `char *cities[10];`
2. `char (*cities)[10];`
3. `char *(cities[10]);`
4. `char *(*cities)[10];`
5. `double (*lines[5])[2];`
6. `double (*arr)[5][2];`
7. `char *city = "Busan";`
8. `const char *city = "Busan";`
9. `char *const city = "Busan";`
10. `const char *const city = "Busan";`
11. `float *g(), (*h)();`
12. `void ex(int, double (*)(int));`
13. `void (*p)(float,float,float (*)(float,float));`
14. `int (*(*p)())();`
15. `int (*func[5])();`
16. `char * const * const p;`
17. `const char * const * const p;`
18. `void *p = new (double (*)[3]);`
19. `int atexit(void (*)(void));`
20. `void (*signal(int, void (*)(int)))(int);`

다음의 설명에 따라 함수 원형을 작성하라.

1. func1 이라는 함수는 전달 인자도 없고, 리턴 값도 없다.
2. func2 라는 함수는 double 전달 인자를 하나 받아서 int 형을 반환한다.
3. func3 라는 함수는 int 참조를 전달 인자로 받아서 char *를 반환한다.
4. func4 라는 함수는 첫번째 인자로 구조체 Arff의 참조, 두번째 인자로 구조체 Map의 참조, 세번째 인자로 int를 받아서, 구조체 Arff의 참조를 반환한다. func4는 첫번째 인자와 두번째 인자의 값을 변경하지 않는다.
5. func5 라는 함수는 문자열 전달 인자를 받아서 int 형을 반환한다. 단, func5 함수는 전달받은 문자열을 변경하지 않는다.
6. func6 라는 함수는 위의 func4 라는 함수의 포인터를 첫번째 인자로 받고, double을 두번째 인자로 받는다. 반환하는 값은 없다.
7. func7 이라는 함수는 위의 func4 라는 함수의 포인터를 첫번째 인자로 받고, 위의 func6 라는 함수의 포인터를 두번째 인자로 받고, 위의 func5 라는 함수의 포인터를 반환한다.

2.3.2 코드 읽기

1. 다음의 코드가 의미하는 바를 설명하라.

```
void main()
{
 (*(void(*)())0)();
}
```

2. 다음의 코드가 의미하는 바를 설명하라. 실행하면 어떤 것을 출력하는가?

```
#include<cstdio>
int main()
{char*s="#include<cstdio>%cint main()%c{char*s=%c%s%c;printf(s,10,10,34,s,34,10);}%c";printf(s,10,10,34,s,34,10);}
```

3. 다음의 코드가 의미하는 바를 설명하라. 실행하면 어떤 것을 출력하고, 어째서 에러가 나지 않고 제대로 출력되는지 설명하라.

```
#include <cstdio>
int main()
{
  (*****printf)("Dongseo U.\n");
}
```

4. 다음의 코드가 의미하는 바를 설명하라. 실행하면 어떤 것을 출력하고, 왜 그런 출력이 나오는지 설명하라.

```
#include <iostream>
void function() { }
int main()
{
  void (*pF)() = function;
  std::cout << pF << std::endl;
  std::cout << *pF << std::endl;
  std::cout << **pF << std::endl;
  std::cout << ***pF << std::endl;
  std::cout << ****pF << std::endl;
  std::cout << *****pF << std::endl;
}
```

5. 다음의 코드가 의미하는 바를 설명하라. 실행하면 어떤 것을 출력하고, 왜 그런 출력이 나오는지 설명하라.

```
#include <iostream>
int main()
{
  double buf[10] = {0,1,2,3,4,5,6,7,8,9};
  double *pb = &buf[-1];
  for (int i=1;i<=10;i++) std::cout << pb[i] << ',';
  std::cout << std::endl;
}
```

6. 다음의 코드가 의미하는 바를 설명하라. 다음 코드의 문제점은 무엇인가?

```
#define SWAP(a,b) (a+=b, b=a-b, a-=b)
```

   (a) 그렇다면 다음 코드의 문제점은 무엇인가?

```
inline void swap(int& a, int& b) {a+=b; b=a-b; a-=b;}
```

   (b) 위의 방법 외에도 XOR을 이용해서 임시 변수 없이 swap을 할 수 있다. 앞의 코드들이 가지는 문제점들을 최대한 해결한 XOR 버전의 swap 인라인 함수를 작성하라.

   (c) 이번에는 두 개의 string 클래스 객체를 고려해 보자. 이 두 개의 string 객체를 임시 변수를 사용하지 않고 swap 하는 함수를 작성하라. (힌트: string 객체의 멤버 함수인 substr 함수 [1]와 length 함수 [2]를 사용하라.)

7. 다음의 코드는 어떤 것을 출력하고, 왜 그런 출력이 나오는지 설명하라.

```
#include <iostream>
int main()
```

[1]substr 함수의 사용법은 http://www.cppreference.com/cppstring/substr.html 를 참조하라.
[2]length 함수의 사용법은 http://www.cppreference.com/cppstring/length.html 를 참조하라.

```
{
  for (int i=0;i<7;i++) std::cout << "Dongseo"[i];
  std::cout << std::endl;
  for (int i=0;i<7;i++) std::cout << i["Dongseo"];
  std::cout << std::endl;
  int nums[7] = {1,2,3,4,5,6,7};
  for (int i=0;i<7;i++) std::cout << i[nums];
  std::cout << std::endl;
}
```

8. 다음의 코드는 어떤 것을 출력하고, 왜 그런 출력이 나오는지 설명하라.

```
#include <iostream>
using namespace std;
int main()
{
  long int money;
  money = 1,000,000; // 백만원!
  cout << money << endl;
  return 0;
}
```

2.3.3 단답형

1. 다음 중 C/C++의 키워드인 것을 모두 골라라.

```
if  printf  main  malloc  cout  sizeof
```

2. 다음의 함수 원형(prototype)가 있다고 하자.

```
void kick(char * name, int times);
```

   (a) times가 디폴트 값으로 1을 갖도록 하려면 어떻게 해야 하는가?
   (b) 함수의 정의는 어떻게 수정해야 하는가?
   (c) name 에 "Chuck Norris"를 디폴트 값으로 갖도록 할 수 있는가? 그렇다면 어떻게 수정해야 하는가?

3. 두 개의 전달 인자들 중에서 더 큰 것을 반환하는 함수 템플릿을 작성하라.

   (a) 전달 인자들이 double 형인 경우에 대한 명시적 구체화를 선언하라.
   (b) 다음과 같은 구조체가 있다고 가정하자.

```
struct Personnel
{
  char name[100];
  double salary;
}
```

이 구조체의 객체 두 개의 참조들를 전달 인자로 받고, salary 멤버 즉 봉급이 더 큰 객체의 참조를 반환하게 하고 싶다. 이를 위한 명시적 특수화 선언과 정의를 작성하라.

4. 함수의 전달 인자가 기본 데이터 형일 때, const 제한자를 사용하지 않는 이유는 무엇인가?

5. 구조체 Student 의 변수 student 가 있다. 이것을 func 라는 아무 것도 반환하지 않는 함수의 하나 뿐인 전달 인자로써 값으로 전달하려면 어떻게 해야 하는가? 그렇지 않고 참조로 전달하려면 어떻게 해야 하는가? 이 두 가지 경우에 대해 함수 선언들을 작성하라.

2.4 프로그래밍 문제 숙제 (소스 파일들을 하나의 ZIP으로 제출)

교재에 나온 다음의 프로그램들을 풀어서 제출하라. 각각 점수는 5점씩이며, 프로그래밍 해답을 보고 푼 경우가 의심되면 10% 감점된다.

- 7.4, 7.6, 7.9, 8.2, 8.6, 8.7

다음의 여덟 개의 프로그래밍 문제들을 풀어서 제출한다.

2.4.1 파이 근사값 계산

우선 파이(π)라는 값에 대해서 알아보자. 파이는 단순히 원의 둘레를 원의 지름으로 나눈 값이다. 그러나 이 값은 먼 옛날부터 현재까지도 많은 사람들을 매료시켰다.

현재까지 알려진 컴퓨터나 다른 방법에 의한 파이의 근사값을 계산하는 공식들은 다음의 것들을 비롯하여 여러 가지가 있다. [3]

- 아르키메데스 (Archimedes) — 원에 내접하는 정다각형의 넓이를 원의 넓이로 근사한 값을 손으로 측정하여 구함 → (컴퓨터 시뮬레이션의 경우 파이 값을 미리 넣어야 함)

$$\pi = \lim_{n=1}^{\infty} \frac{n}{2} \sin \frac{2\pi}{n}$$

- 비에트 (Viete)

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots$$

- 월리스 (Wallis)

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdots$$

[3] 더 자세한 내용은 페트로 베크만의 "π의 역사"나 "http://ko.wikipedia.org/wiki/원주율"과 "http://en.wikipedia.org/wiki/Pi" 을 참조하라.

- 뉴턴 (Newton)

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2}\left(\frac{1}{3 \times 2^3}\right) + \frac{1}{2} \times \frac{3}{4}\left(\frac{1}{5 \times 2^5}\right) + \frac{1}{2} \times \frac{3}{4} \times \frac{5}{6}\left(\frac{1}{7 \times 2^7}\right) \cdots$$

- 라이프니츠 (Leibniz)

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \cdots$$

- 샤프 (Sharp)

$$\pi = 2\sqrt{3} \times \left[1 - \left(\frac{1}{3 \times 3}\right) + \left(\frac{1}{3^2 \times 5}\right) - \left(\frac{1}{3^3 \times 7}\right) + \cdots\right]$$

- 마친 (Machin)

$$\frac{\pi}{4} = 4\left(\frac{1}{5} - \frac{1}{3 \times 5^3} + \frac{1}{5 \times 5^5} - \cdots\right) - \left(\frac{1}{239} - \frac{1}{3 \times 239^3} + \frac{1}{5 \times 239^5} - \cdots\right)$$

- 체비쉐프 (Chebyshev)

$$\frac{\pi}{8} = \sum_{n=0}^{\infty} \frac{(-1)^n \left(\sqrt{2} - 1\right)^{2n+1}}{2n+1}$$

$$\frac{\pi}{12} = \sum_{n=0}^{\infty} \frac{(-1)^n \left(2 - \sqrt{3}\right)^{2n+1}}{2n+1}$$

- 손도우 (Sondow)

$$\pi = \frac{\prod_{n=1}^{\infty}\left(1 + \frac{1}{4n^2-1}\right)}{\sum_{n=1}^{\infty}\left(\frac{1}{4n^2-1}\right)}$$

- 오일러 (Euler) - 리만 제타 함수

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} \cdots$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} \cdots$$

- 뷔퐁(Buffon)의 바늘 실험 — 몬테카를로 방법 (l:바늘의 길이, t(>l):라인들 간의 거리, n:떨어뜨린 바늘들의 수, h:라인에 교차한 바늘들의 수) → (컴퓨터 시뮬레이션의 경우 파이 값을 미리 넣어야 함)

$$\pi = \frac{2ln}{th}$$

- 단위 원(circle)의 면적(π)과 그 단위 원에 외접하는 사각형(square)의 면적($4 = 2^2$)의 비율을 이용하는 몬테카를로 방법

$$\pi = \frac{4 * (\#\text{ of points in the circle})}{(\#\text{ of points in the square})}$$

- 라마누잔(Ramanujan), 조나단 보와인과 피터 보와인

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{n=0}^{\infty} \left(\frac{(4n)!}{(n!)4} \times \frac{[1103 + 26390n]}{(4 \times 99)^{4n}} \right)$$

- 연분수(continued fraction)

$$\frac{4}{\pi} = 1 + \cfrac{1}{3 + \cfrac{4}{5 + \cfrac{9}{7 + \cfrac{16}{9 + \cfrac{25}{11 + \cfrac{36}{13 + \ldots}}}}}}$$

$$\pi + 3 = 6 + \cfrac{1}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \cfrac{11^2}{6 + \ldots}}}}}}$$

여러분의 이해를 돕기 위해 월리스 (Wallis)의 수식을 이용하여 파이를 구하는 프로그램을 예로써 제시하겠다.

```
#include <iostream>
int main()
{
  int iteration = 0;
  std::cout << "Enter # of iterations :"; // 1000000
  std::cin >> iteration;
  long double pi = 1.0L;
  for (int i=1;i<=iteration;i++)
  {
    long double odd = i;
    long double even = i;
    if (0==(i%2)) odd++; else even++;
    pi*=(even/odd);
  }
  pi=2.0L*pi;
```

```
  std::cout.precision(18);
  std::cout << pi << std::endl;
}
```

위에서 언급한 수학식들 중 아르키메데스의 방법과 뷔퐁의 바늘 실험, 그리고 월리스의 방법을 제외하고 3 가지 방법을 골라 파이의 값을 구하는 프로그램들을 작성하라. 즉, 위에서 아르키메데스의 방법과 뷔퐁의 바늘 실험, 그리고 월리스의 방법을 제외하고 3 개의 방법을 찾아서, 각각에 대해 프로그램을 구현하는 것이다. 결과적으로 3 개의 서로 다른 프로그램들을 만들어서 제출하게 될 것이다. 이 문제의 점수는 20점이다.

2.4.2 셀프 넘버(self-number)

이 문제의 점수는 20점이다.

어떤 자연수 n이 있을 때, $d(n)$을 n의 각 자릿수 숫자들과 n 자신을 더한 숫자라고 정의하자. 예를 들어

$$d(91) = 9 + 1 + 91 = 101$$

이 때, n을 $d(n)$의 제네레이터(generator)라고 한다. 위의 예에서 91은 101의 제네레이터이다. 어떤 숫자들은 하나 이상의 제네레이터를 가지고 있는데, 101의 제네레이터는 91 뿐 아니라 100도 있다. 그런데 반대로, 제네레이터가 없는 숫자들도 있으며, 이런 숫자를 인도의 수학자 Kaprekar가 셀프 넘버(self-number)라 이름 붙였다. 예를 들어 1,3,5,7,9,20,31 은 셀프 넘버 들이다.

이제 문제를 내겠다. 1 이상이고 5000 보다 작은 모든 셀프 넘버들의 합을 구하는 프로그램을 작성하라.

이 문제는 게임 회사인 넥슨의 입사 문제라고 알려진 문제들 중 제일 쉬운 것이다.

2.4.3 3n+1

이 문제의 점수는 10점이다.

다음과 같은 함수를 생각해 보자.

```
void ThreeNPlusOne(int n)
{
  while (1<n)
  {
    cout << n << " ";
    if (0==(n%2)) n=n/2; else n=3*n+1;
  }
  if (1==n) cout << n;
  cout << endl;
}
```

만일 n 이 22 이면, 위의 함수를 실행하면 다음과 같이 출력되고 끝난다.

```
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

많은 수학자들이 모든 정수에 대해 저 알고리즘이 결국 1로 멈추는지를 증명하고자 애쓰고 있다. 이 문제를 3n+1, 또는 Collatz 문제라고 부른다. 현재까지는 상당히 큰 수(10×2^{58})까지에 대해서 저 알고리즘이 멈춘다는 사실이 알려져 있을 뿐이다.

위의 함수를 조금 바꾸면 주어진 n에 대해 얼마나 많은 숫자가 출력되는지를 구할 수 있다. 예를 들어 위의 22 의 경우, 16 개의 숫자가 출력되었다. n이 6인 경우를 보면, 출력 결과는 다음과 같이, 9 개의 숫자가 출력된다.

```
6, 3, 10, 5, 16, 8, 4, 2, 1
```

n 이 11인 경우, 다음과 같이 15 개의 숫자가 출력된다.

```
11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

n 이 27 인 경우, 112 개의 숫자가 출력된다. 이 경우는 직접 돌려서 확인해 보기 바란다. 이렇게 출력되는 숫자들의 개수를 cycle length라고 부른다.

이제 문제를 내도록 하겠다. 일단 두 개의 정수를 입력받는다. 입력받은 정수들은 0보다 크다고 가정한다. 그 두 개의 정수 사이에 있는 모든 정수들(입력받은 두 개의 정수 포함)에 대해 위의 3n+1 알고리즘을 수행하여, 그 중 가장 많은 숫자가 출력되는 정수를 찾아 그 정수와 출력되는 숫자들의 개수를 출력하는 프로그램을 작성하라.

예를 들어, 입력된 정수가 1과 10이면, 가장 많은 숫자가 출력되는 정수로 9가 그리고 그 출력되는 숫자의 개수로 20이 출력되어야 한다.

다음은 프로그램의 수행 예이다.

```
Enter starting number :1
Enter ending number :10
1
2 1
3 10 5 16 8 4 2 1
4 2 1
5 16 8 4 2 1
6 3 10 5 16 8 4 2 1
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
8 4 2 1
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
10 5 16 8 4 2 1
9 with maximum length 20.
```

2.4.4 4 개의 말뚝을 가진 하노이의 탑

하노이의 탑은 재귀 호출을 이용한 테스트 프로그램 중 하나로 잘 쓰인다. 하노이의 탑 문제는 간단하다.

말뚝이 3 개 있고, 그 말뚝(peg)에 꽂을 수 있는 접시의 개수는 사용자에게 입력받는다. 처음에는, 접시는 길이가 작은 것이 길이가 큰 것보다 위로 가도록, 모두 하나의 말뚝(주로 맨 왼쪽의 말뚝)에만 꽂혀있다. 이 문제의 목표는 하나의 말뚝에만 있는 접시들을 다른 하나의 말뚝으로 모두 옮기는 것이다. 단 두 가지 제한 사항이 있다.

- 한 번에 하나의 접시만 옮긴다.
- 어떠한 경우에도, 즉 옮기는 도중이나, 처음, 또는 마지막, 어떠한 경우에도, 큰 접시가 자기보다 작은 접시보다 위에 있으면 안된다.

하노이의 탑 알고리즘의 근본 원리는 다음과 같다. 일단 3 개의 말뚝들을 처음에 접시들이 있는 시작 말뚝, 접시들을 옮길 목표 말뚝, 그리고 중간에 사용할 말뚝으로 각각 나눈다. 알고리즘은 n-1 개의 디스크를 중간 말뚝으로 옮기고, 남은 하나의 디스크를 목표가 되는 말뚝으로 옮긴 후, 앞에서 중간 말뚝으로 옮긴 n-1 개의 디스크를 다시 목표 말뚝으로 옮긴다는 것 뿐이다. 이를 구현하면 다음과 같다.

```
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

void hanoi(int fromPeg, int toPeg, int usingPeg, int n)
{
  if (n>1) hanoi(fromPeg, usingPeg, toPeg, n-1);
  cout << fromPeg << "->" << toPeg << endl;
  if (n>1) hanoi(usingPeg, toPeg, fromPeg, n-1);
}

int main()
{
  cout << "Input the number of disks :";  // 접시의 개수를 입력받는다.
  int n = 0;
  cin >> n;
  hanoi(1,2,3,n);
  return 0;
}
```

이제 문제를 내겠다. 이 문제의 점수는 20점이다. 문제는 말뚝의 개수가 3이 아니라 4인 경우에 대해 프로그램을 구현하는 것이다. (더 나아가 3이나 4 또는 5같은 정해진 수가 아닌 임의의 m 개의 말뚝을 입력받으면 어떻게 해야 하는가라는 문제가 있으나, 난이도가 상당히 높으므로 여기선 말뚝이 4 개인 경우로만 문제를 제한하겠다. 그러나, 자신있는 학생은 m 개의 말뚝에 대해 도전해서 결과를 내면 보너스 점수를 받을 것이다.)

사실, 위의 프로그램을 제대로 이해했다면, 말뚝이 4개인 경우도 단순하게 푼다면 문제는 그다지 어렵지 않다. 다만, 알고리즘을 어떻게 구성하느냐에 따라 필요 이상으로 더 많이 접시를 움직여서 문제를 해결할 수도 있다. 3 개의 말뚝을 사용하는 원래 하노이의 탑 문제의 경우, n 개의 접시를 입력한 경우, $2^n - 1$ 번 접시를 옮기면 전체가 옮겨진다는 사실이 현재 증명되어 있다. 그런데, 4 개의 말뚝이나 m 개의 말뚝을 사용하는 경우, n 개의 접시들을 옮길 수 있는 최소의 개수는 현재 증명되지 않은 상황이다. 다만 이 최소의 개수가

Frame–Stewart Conjecture (프레임이라는 사람과 스튜어트라는 사람의 추측)를 따를 거라는 추측을 하고 있을 뿐이다.

따라서, 본 문제에서는 학생들이 구현한 알고리즘이 최소의 개수로 옮기는 것을 요구하지 않는다. 단지 제대로 옮기기만 하면 된다. 대신 가장 작은 개수로 옮기는 알고리즘을 구현한 학생에게는 보너스 점수를 주도록 하겠다.