



Object-Oriented Programming Report

Assignment 1-1

Professor	Donggyu Sim
Department	Computer engineering
Student ID	2022202065
Name	박나림
Class (Design / Laboratory)	2 / C
Submission Date	2023. 3. 15

Program 1

□ 문제 설명

사용자로부터 unsigned char 타입의 변수 $k(1 \leq k \leq 8)$ 를 입력 받아서 한 변의 길이가 $N(2^{k-1} \cdot 3)$ 인 시에르핀스키 삼각형을 출력하는 프로그램이다. 이때 출력 기호는 '\$'를 사용한다.

-unsigned char: 음수 표현의 문제를 방지하기 위해 양수로만 나타내는 타입이다.
 $0 \sim 2^8(255)$ 크기의 숫자를 나타낼 수 있다.

-시에르핀스키 삼각형: 정삼각형 하나에서 시작하여 세 변의 각 중점을 이으면 안에 또 하나의 작은 정삼각형이 만들어진다. 그 작은 정삼각형을 제거한 후 남은 정삼각형들도 똑같은 방법을 무한히 계속 반복하면 나오는 형태가 시에르핀스키 삼각형이다.

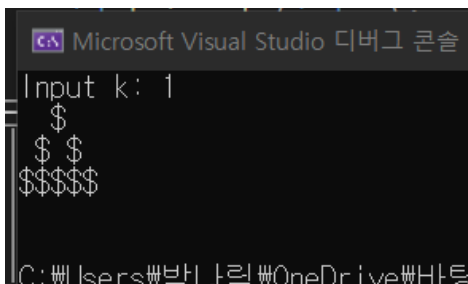
- 구현 방법

먼저 int 형 변수 k, N 을 선언한다. 그리고 while 문을 통해 사용자로부터 정해진 범위에 대해서만 k 값을 받도록 하는데, 범위에서 벗어날 시 다시 입력 받도록 만든다. 제대로 된 값을 입력하면 while 문을 벗어나서 N 을 구하도록 한다. 이때 거듭제곱 계산은 <cmath>의 pow()를 이용한다. 그리고나서 시에르핀스키 삼각형을 만들 함수 Triangle(int n)을 호출한다.

Triangle 함수에서는 먼저 char 형 2 차원 배열 tri 를 선언하여 삼각형 모양으로 초기화 시킨다. 그리고 for 문을 이용하여 전달받은 N 값에 따라 삼각형을 출력한다.

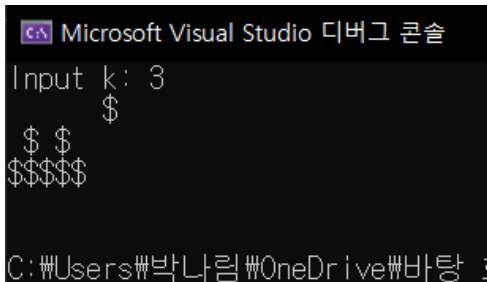
□ 결과 화면

1) 1 을 입력했을 때 제대로 작동됨



```
Microsoft Visual Studio 디버그 콘솔
Input k: 1
$
$$
$$$
$$$$
C:\Users\박나림\OneDrive\바탕
```

2) 2 이상부터는 해결 못함



```
Microsoft Visual Studio 디버그 콘솔
Input k: 3
$
$ $
$ $ $
$ $ $ $
$ $ $ $ $
C:\Users\박나림\OneDrive\바탕 화면\
```

□ 고찰

이 문제에서 제일 큰 핵심이 시에르핀스키 삼각형을 만드는 것인데 결과적으로 완성하지 못하였다. main 함수에서 입력 받은 값을 통해 N 값을 구하고 함수로 넘기는 것까지는 구현했지만 그 함수부터가 문제였다. 여러가지 방법을 고민해봤지만 삼각형을 배열로 저장했을 때 첫번째 행의 \$ 기호가 N 값이 2 이상이 될 때 두번째 삼각형의 \$도 같이 첫번째 행부터 들어가야 한다는 점이 어려웠다. 그렇게 같은 행에 들어가기 위해서는 기존처럼 한번에 배열로 삼각형을 저장하는 게 아니라 \$기호 각각을 따로따로 저장해야 될 것 같다. 또한 그냥 반복문을 사용하기 보다 같은 모양이 반복되니 재귀함수를 호출하는 게 효율적일 것 같아 보였다.

전체적인 흐름은 이해했지만 직접 각 변의 중점에 가서 그리는 형식이 아니라 배열을 통하여 구현하려니 생각보다 더 어려웠다. 해결방법으로 떠올린 생각들을 더 구체화시켜서 해봐야 할 것 같다.

Program 2

□ 문제 설명

사용자로부터 실수 변수(float data type) 3 개를 입력 받아서 차례대로 a, b, c 라 할 때 이를 이용하여 근의 공식을 계산하는 프로그램이다.

$$\text{-근의 공식: } x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (a \neq 0)$$

이때 결과 값을 부호대로 X1, X2 로 나누어서 출력한다. 분모가 0 이 될 경우에는 식 자체가 성립하지 않으므로 안내문을 출력하며 종료하고, 판별식($b^2 - 4ac$)이 0 이 되는 경우에는 값과 함께 중근(double root)안내문을, 0 보다 작은 경우에는 허근이라는 안내문을 출력하고 종료한다.

하지만 기존의 근의 공식을 그대로 사용할 경우 실제 값과 다르게 오차가 발생한다. float 타입(4byte=32bit)을 사용할 시 소수점 6 자리까지는 정확히 나오지만 그 외에는 부동소수점 방식으로, 실수에서 소수점을 표현할 때 2 진수로 바뀌면서 근삿값이 저장되는데 그 때 소수점이 정확하게 떨어지지 않는 무한소수가 되는 문제가 발생하기 때문이다. 따라서 위처럼 근의 공식을 계산할 때 유사한 두 숫자 간 뺄셈이 들어간 경우 에러가 더 커질 수 있다는 것이다. 그리하여 식을 변형하여 사용하면 에러를 줄일 수 있다.

- 구현 방법

먼저 사용자로부터 입력 받을 3 개의 값 a, b, c 를 float 타입으로 선언해준다. 그리고 차례대로 입력을 받고 값을 저장한다. 그 다음 연산 차례에서, 먼저 식이 성립되는지 확인하기 위해 if 조건문으로 분모에 들어가는 a 의 값이 0 인지 확인한다. 만약 0 이라면 안내문을 띄우고 바로 종료시켜서 프로그램이 그 다음에 하는 연산 진행을 의미 없이 하지 않도록 만든다.

그 다음 연산 차례로, float 타입 D 변수를 선언하여 판별식을 저장한다. 같은 타입으로 식의 결과 값을 연산하여 저장할 변수들도 선언해준다. 이때 위에서 말한 오차 에러를 줄이기 위하여 변형한 식을 저장하는데, b 의 값이 양수인지 음수인지에 따라 변수를 X1, X2 와 x1, x2 로 나눈다. 양수일 경우에는 $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ 인 첫번째 근 X1 을, 음수일 경우에는 $\frac{-b - \sqrt{b^2 - 4ac}}{2a}$ 인 두번째 근 x2 를 변형하도록 한다.

첫번째 근에서는 각 분모, 분자에 $(-b - \sqrt{b^2 - 4ac})$ 를 곱하여 $\frac{2c}{-b - \sqrt{b^2 - 4ac}}$ 가 되도록 만든다. 여기서 두번째 근의 식은 그대로 진행한다. 음수일 경우에는 이와 반대로 진행하여 식을 변형한다.

마지막으로 조건문을 이용하여 if, else if, else 로 차례대로 허근일 경우에 안내문을, 중근을 가질 경우에 값과 함께 안내문을, 서로 다른 실근을 가질 경우에는 또 다시 if 문으로 b의 양수 음수 여부에 따라 출력 변수를 달리하여 각각의 값들을 출력하고 종료한다.

□ 결과 화면

1) 분모가 0 이 될 때 안내문 띄우고 종료

```
Microsoft Visual Studio 디버그 콘솔
a: 0
b: 1
c: 1
Unexpected factor of a quadratic term
C:\Users\박나림\OneDrive\바탕 화면\과제\
다(코드: 1개)
```

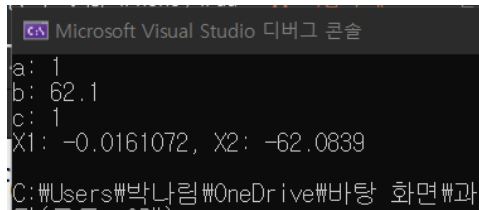
2) 중근이 될 때 값 출력과 함께 중근임을 알림

```
Microsoft Visual Studio 디버그 콘솔
a: 1
b: -2
c: 1
X1, X2: 1(double root)
C:\Users\박나림\OneDrive\바탕 화면\
```

3) 허근이 될 때 안내문 띄우고 종료

```
Microsoft Visual Studio 디버그 콘솔
a: 1
b: 1
c: 1
The equation has no real number solutions.
C:\Users\박나림\OneDrive\바탕 화면\과제\Assign
ment\code\
```

4) 서로 다른 실근이 될 때 각각의 값 출력



```
Microsoft Visual Studio 디버그 콘솔
a: 1
b: 62.1
c: 1
X1: -0.0161072, X2: -62.0839
C:\Users\박나림\OneDrive\바탕 화면\과
```

□ 고찰

이번 문제에서는 근의 공식을 변형하는 게 제일 중요한 포인트였다. 사용자가 입력한 값들이 기존의 근의 공식으로 들어갈 때 특정 케이스에서는 분자의 위치에서 유사한 두 숫자 간 뺄셈 연산이 되어버린다. 그렇게 되면 float 타입에서 표현할 수 있는 비트를 넘어갈 경우 반올림으로 인해 오차가 발생한다. 따라서 공식을 적절히 변형하여서 그러한 뺄셈 연산을 피하도록 만들어야 되는데, 처음에는 어떤 식으로 변형할지 잘 모르겠어서 여러 시행착오들을 겪었다.

여러가지 방향으로 변형을 생각해보던 중, 분자에 있는 식이 문제이니 분모로 옮겨보자는 생각이 들어 위에 말했던 방식으로 두개의 근 모두 변형시켜봤다. 그렇게 하였더니 주어진 문제의 4 번째 케이스를 진행할 때 X1 값이 제대로 나오는 걸 확인하였다. 하지만 이번엔 X2 값이 달라져서 정확히 나오지 않았다. 이에 X2 식은 기존의 공식을 다시 그대로 활용하였고, 다른 케이스도 진행하다 보니 b 값의 양수 음수 여부에 따라 서로 변형식을 반대로 진행해야 된다는 걸 알게 되었다.

그렇게 다시 해본 케이스들에 대해서는 성공했지만, 이 변형식이 어떠한 원리로 작동되는 것인지 아직 완전히 이해하지 못한 점이 아쉽다. 이 부분은 비트와 2 진수 체계에 대하여 더 공부해 볼 것이다.

Program 3

□ 문제 설명

GCD(최대 공약수) 함수를 이용하여 LCM(최소 공배수)를 구하는 프로그램으로, 내장된 GCD() 함수를 사용하는 게 아니라 직접 구현해야 하는 게 조건이다. int 형 변수 x, y를 선언하여 x는 gcd를 위해 가장 큰 수가 들어가고, y는 작은 수가 들어가서 나머지를 남지 않을 때까지 계속 나누는 역할을 하게 한다. 즉, 나머지를 구하는 식인 $x \% y$ 를 연산하여 나온 결과 값을 다시 y에 저장하여 나머지가 0이 될 때까지 반복한다는 뜻이다. gcd(x, y) 함수를 재귀함수로 설정하여 다시 호출할 시 gcd(y, x%y)가 되게 한다. 그리하여 y가 0이 되는 순간의 x값이 최대 공약수가 된다. 그 값을 이용하여 마지막으로 최소 공배수를 구하면 되는 문제이다.

-구하는 식: $LCM = (X*Y) / GCD$

단, 이때 연산하는 data type이 int형(4byte)이므로, 연산 결과가 32bit를 넘어가면 bit가 소멸되는 오버플로우가 발생할 수 있다는 문제점이 있다. 따라서 변수를 입력 받을 때 최대한 오버플로우를 피하도록 해야 한다.

- 구현 방법

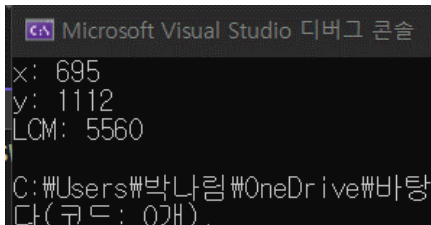
사용자로부터 두 수를 입력 받기 위해 변수 x, y를 선언하여 차례대로 값을 입력 받아 저장한다. 그 다음 연산을 진행하기 위해 둘 중 더 큰 수를 x에 놓도록 if문을 사용하여 만약 y값이 더 클 시 swap 함수를 호출하여 값을 교환하도록 만든다. swap 함수는 main 위에 추가하여 값이 제대로 교환되도록 포인터를 이용하여 주소 값을 받도록 한다.

값들이 정렬되고 나면 LCM 변수를 선언하여 최소 공배수를 구하는 식을 저장한다. 이때 필요한 값인 최대공약수는 함수 자체를 식에 넣어서 만든다. gcd 함수도 마찬가지로 main 위에 추가하여 만드는데, 원하는 값을 구하기 위해서는 계속 반복해서 연산을 진행해야 하므로 재귀함수 형식으로 만들도록 한다. if문을 사용하여 y가 0이 되면 int 타입인 x값을 반환하도록 하고, y가 0이 되기 전까지는 자기 자신 함수를 다시 호출하게 한다. 이때 큰 틀의 함수는 gcd(int x, int y)로 하고, 재귀적으로 다시 호출할 때에는 gcd(y, x%y)로 하여서 연산을 진행하게 한다.

최종적으로 반환된 값으로 LCM을 구하고 나면 그 값을 출력하면서 끝낸다.

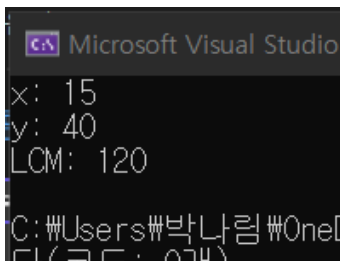
□ 결과 화면

- 1) 695 와 1112 를 입력했을 때, 연산을 위해 x, y 값을 교환하고 $\text{gcd}()$ 를 계산하여 LCM 까지 구한 결과 5560



```
Microsoft Visual Studio 디버그 콘솔
x: 695
y: 1112
LCM: 5560
C:\Users\박나림\OneDrive\바탕
다(코드: 0개)
```

- 2) 15 와 40 를 입력했을 때, 연산을 위해 x, y 값을 교환하고 $\text{gcd}()$ 를 계산하여 LCM 까지 구한 결과 120



```
Microsoft Visual Studio
x: 15
y: 40
LCM: 120
C:\Users\박나림\OneD
다(코드: 0개)
```

□ 고찰

위의 문제에서 마지막으로 말한 주의사항인, 오버플로우를 방지하기 위해 입력 받을 때 변수의 범위를 지정해줘야 하는 점을 완전히 보완하지 못했다. 연산을 진행하다 보니 큰 수를 입력 받더라도 마지막엔 그만큼 커진 GCD로 값을 나눠주기 때문에 결과값이 제대로 나왔다. 하지만 더욱 큰 수를 입력하는 케이스의 경우에는 앞서 말한 오버플로우 문제가 일어날 것 같다. 그렇다면 역시 범위를 지정해주는 게 안전할 것이다. 하지만 단순히 처음 x, y 의 int 형 비트로 계산하여 $-2^{31} \sim 2^{31}-1$ 범위로 하기에는 연산 결과에 따라 또 다시 값의 크기가 달라지므로 다른 범위를 생각해야 할 것 같다. 이 점 역시 바이너리 시스템에 대해 더 공부해봐야 될 것 같다.

Program 4

□ 문제 설명

사용자로부터 정수 9 개를 입력 받아서 3x3 크기의 역행렬을 구하는 프로그램이다. 이때 결과 값은 double 형으로 나오게 한다. 또한 역행렬의 성립 조건을 고려해야 하는데, 행렬식 $\det A$ 가 0 이 되면 역행렬이 될 수 없으므로 그러한 경우에는 역행렬이 존재하지 않는다는 안내문을 띄우고 프로그램을 종료시키도록 한다.

-역행렬 공식: $A^{-1} = \frac{1}{\det A} C^T$ (행렬 A, 여인수행렬 C 의 전치행렬 C^T)

- 구현 방법

먼저 배열 값 계산을 위한 기호들로, a~i 로 9 개의 변수들을 int 형으로 선언해준다. 그리고 역행렬 조건 판단을 위한 double 형 det_A, 사용자로부터 입력 받을 배열로 int 형 2 차원 배열 A, 마찬가지로 타입으로 여인수행렬 C, 수반행렬(전치) C_T 까지 선언한다.

그 다음 이중 for 문을 이용하여 사용자로부터 3 by 3 크기의 배열 값들을 입력 받는다. 이 값들을 편하게 계산하기 위해 처음에 선언해 놓은 변수 9 개로 치환하여 저장한다. 이후 det_A 식을 계산하고, if 문을 이용해 det_A 가 0 이 될 경우 역행렬이 존재하지 않는다는 안내문을 띄우고 프로그램을 종료시켜서 의미 없는 연산을 진행하지 않도록 만든다.

역행렬 성립 확인까지 끝나면 연산 과정 진행으로, 먼저 여인수행렬을 구하기 위해 C 배열에 각각 계산 식들을 저장해준다. 그리고 C 의 전치행렬을 구하고 역행렬 최종 공식에 대입하기 위해 det_A 의 역수까지 곱하여 구하는 과정을 이중 for 문을 이용하여 한번에 처리할 수 있도록 만든다. 마지막으로 다시 반복문을 이용하여 역행렬을 3 by 3 모양으로 나오도록 출력해주고 끝낸다.

□ 결과 화면

- 1) $\det A$ 가 0 이 되어 역행렬이 성립하지 않는 경우 안내문을 띄우고 종료

```
Microsoft Visual Studio 디버그 콘솔
2 2 1
-1 1 0
0 0 0
The inverse matrix does not exist.
C:\Users\박나림\OneDrive\바탕 화면\
```

- 2) 역행렬이 되는 값을 입력할 경우 제대로 연산 과정이 진행되어 역행렬을 구한 결과

```
Microsoft Visual Studio 디버그 콘솔
1 2 3
0 1 4
5 6 0
-24 18 5
20 -15 -4
-5 4 1
C:\Users\박나림\OneDrive\바탕 화면\
```

□ 고찰

이번 문제에서는 다른 특별한 에러들을 해결하는 문제보다 어떻게 프로그램을 더 좋은 방향으로 짤 수 있을지가 제일 고민이었다. 특히 행렬을 계산하는 과정에서 직접 배열 값 하나하나에 전부 식들을 대입하였던 게 아쉬웠다. 다른 방법도 고민해보면서 여러 시행착오들을 겪었었다. 하지만 기호들로 연산할 때 각 배열에 들어가는 값마다 전부 연산식이 달라져서 반복문을 쓰고 싶어도 어떻게 해야 할지 잘 모르겠다. 그래서 처음부터 값을 치환하지 않고 그냥 배열 숫자로서 계산하는 방법도 생각해봤지만 그럼 처음에 $\det A$ 를 구하는 과정에서 복잡해질 것 같아 고치지를 못하였다. 배열 계산용 함수를 따로 만들어서 효율적으로 하는 방법도 시도해봤는데 2차원 배열에 대한 매개변수 전달 방법 등에 대해서도 시행착오가 있었다. 따라서 이번에는 전부 다 대입하는 방식으로 진행하였지만, 앞으로 배열과 함수에 대해 더 공부하여 다른 효과적인 방법이 있는지 더 고민해볼 것이다.