# Object-Oriented Programming Report

Assignment 1-1

| | |
|---|---|
| Professor | Donggyu Sim |
| Department | Computer engineering |
| Student ID | 2022202093 |
| Name | Wonju moon |
| Class (Design / Laboratory) | 1 / B (미수강시 0로 표기) |
| Submission Date | 2023. 3. 17 |

서식 지정함: 글꼴: (영어) 한컴 바탕, (한글) 한컴 바탕, 글꼴 색: 텍스트 1

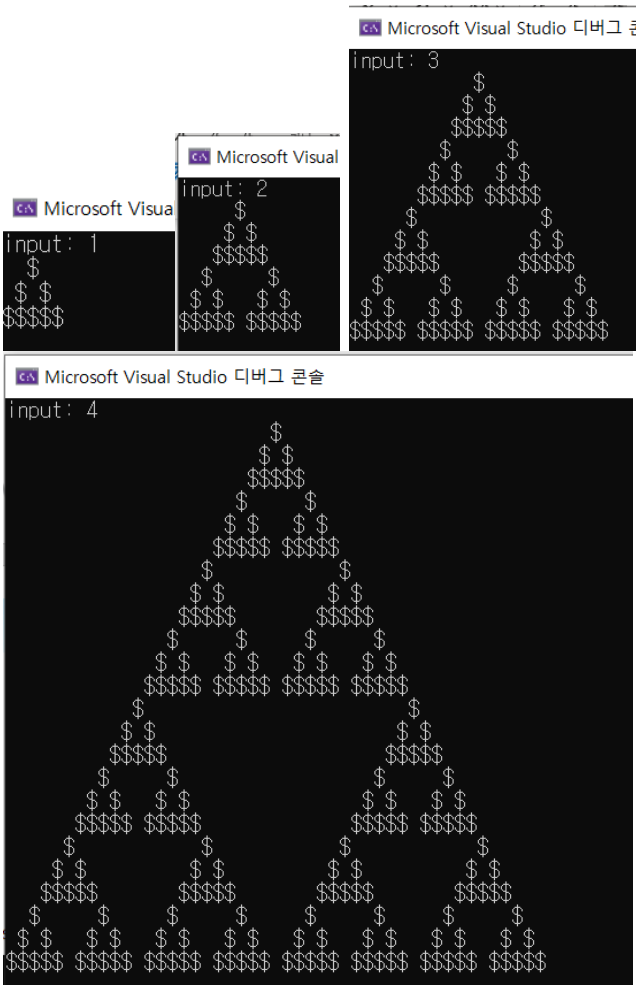
서식 있음: 간격 없음, 줄 간격: 1줄, 눈금에 맞춤, 텍스트 맞춤: 자동

# Program 1

## ❑ 문제 설명

- ❑ **unsigned char형으로 k값을 입력 받아 int형으로 변환을 한 후에 시에르핀스키 삼각형을 완성하라.**
- ❑ 시에르핀스키 삼각형은 기본적으로 프랙탈 구조로 형성된다. 프랙탈 구조에서는 재귀함수가 많이 이용되는데 시에르핀스키 삼각형에서도 마찬가지였다. K의 값이 최대 8까지만 입력 가능하기 때문에 2차원 배열을 전역변수로 선언하여 좌표를 잡고 모든 좌표의 값을 공백으로 초기화 해준다. 그 후 마치 수학에서 좌표를 이용하는 것처럼 해당 위치에 $를 입력하는 작업을 한다. 이 때, 재귀함수를 사용하는데 인자 값으로는 삼각형의 높이인 height와 삼각형 한 변의 길이인 length 그리고 i값이 들어간다. 이때 i란 처음에 높이 값을 받아서 계속 절반씩 나눠가며 height와 length의 값을 수정해준다. length와 height의 값은 k값에 따라 변하는데 결국 좌표적으로 설정하여 i==3이 될 때까지 좌표값을 수정해주고 i==3이 되면 해당 자리에 $를 입력한다. 삼각형 가운데를 기준으로 좌측과 우측을 나눠찍는 과정을 반복하여 프랙탈 구조를 완성한다.

## ❑ 결과 화면




k=2 인 경우를 봐보자. 이때 height 는 6 이고 length 는 11 이다. Fractal 함수로 전달되는 인자 값은 i=6, height =0, length =5 이다. 이때 함수는 재귀적으로 작동하기 때문에 fractal 내에서 다시 i=3 height =0 length=5 가 입력된다.

I=3 조건을 만족시켰기 때문에 삼각형이 하나 그려진다. 그 후 두번째 fractal 이 돌면서 좌표가 바뀐다. 이때는 i=3 과 height=3 length=2 라는 값이 대입되는데 마찬가지로 i=3 이기 때문에 해당

서식 지정함: 글꼴: 10 pt

좌표에 삼각형을 하나 더 그리게 된다. 이렇게 하면 가운데 삼각형의 왼쪽 부분에 삼각형이 하나 더 생기고, 3 번째 fractal 함수에서도 마찬가지로 오른쪽 삼각형이 생기게 된다.

### ❑ 고찰

문제를 처음 2 차원 동적할당을 사용해서 풀려고 시도해봤다. 그러나 2 차원 배열의 동적할당을 함수로 전달할 때 어떻게 전달을 해야하는 지에 어려움을 느꼈다. 그리고 동적할당 때문에 배열의 크기가 자꾸 바뀌어서인지 이상한 언어가 나오고 문양이 제대로 찍히지 않았다. 런타임 오류도 많이 나서 결국 포기하고 배열 값을 고정해둔 상태로 문제를 해결했다. 전역변수로 배열을 선언하니 따로 인자 값을 전달해주거나 복잡하게 배열을 여러 개 선언하지 않아도 해결할 수 있었다. 또한, 재귀함수를 다루는 실력이 많이 부족해서 재귀함수를 짠다는 생각이 어려웠고 많은 시행착오가 있었다. 또한, k 값을 char 형으로 받아서 숫자로 치환하는 것도 아스키코드라는 아이디어가 떠오르지 않아 구글링에 도움을 받았다.

서식 지정함: 글꼴: 10 pt

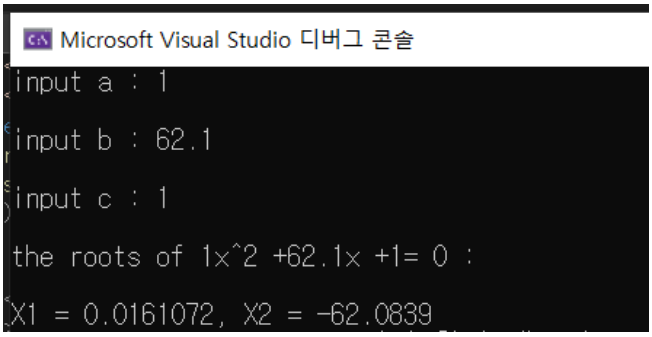
## Program 2

### ❑ 문제 설명

❑ **문제에서 제시된 근의 공식을 사용할 경우 오차가 발생함. 이러한 이유는 근의 공식의 분자 부분에서 유사한 두 숫자 사이의 뺄셈 연산 시 에러가 증폭되는 현상이 발생하기 때문임.**

❑ 근의 공식에서 분자 부분에 있는 뺄셈이 오류를 발생한다고 판단함. 그래서 경우의 수를 생각하여 분자가 뺄셈이 되는 경우 유리화를 진행해서 계산하게 만듦. 유리화를 진행하면 뺄셈으로 표현된 음수가 전부 덧셈으로 바뀌기 때문임. 그러나 오차 값이 여전히 존재함.

❑ 결과 화면




X1 값에서 0.0000004 의 오차 값의 발생을 유리화를 통해 분자의 뺄셈을 없애서 오차를 없애줌.

서식 지정함: 글꼴: 10 pt

❑ 고찰

처음에 함수를 짤 때 Discriminant 라는 함수에서 판별식과 근의 공식을 한번에 판단하는 함수를 짰다. 그러나 왜인지는 모르겠으나 판별식과 근의 공식을 한번에 판단하면 오차가 잡히지 않았다. 그래서 Discriminant 는 실근의 유무를 판단하는 판별식으로 남겨두고 roots 라는 함수를 새로 짜서 판별식과 근의 공식을 따로 판단하는 방식으로 프로그램을 짰다. 또한, pow 함수가 double 형이어서 근의 공식에서 분자에 있는 루트 안 식을 구할 때 pow 함수를 쓰면 오차가 또 발생하는 문제가 생겼다. 그래서 근을 직접 구할 때는 pow 함수를 사용하지 않았다. 그랬더니 잡히지 않은 오차가 잡혔다.

서식 지정함: 글꼴: 10 pt

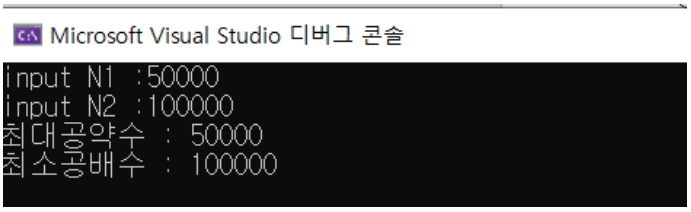
## Program 3

## ❑ 문제 설명

❑ **재귀함수를 이용해 최소공배수와 최대공배수를 구해라. 이 때 주어진 값이 overflow가 발생하는 경우 overflow를 해결하라.**

❑ 결국 유클리드 호제법으로 푸는 최소공배수와 최대공약수 문제. 그러나 최소공배수를 구하는 과정에서 두 수의 곱이 이루어지는데 두 수가 곱해질 때 int형의 범위인 21억을 초과하는 overflow가 발생할 수 있다. 따라서 int형을 long long int형으로 바꿔서 범위를 넓혀줬다.

## ❑ 결과 화면




재귀함수를 통해서 최대공약수를 먼저 구해준다. 유클리드 호제법으로 값을 구하면 아주 간단한데, 큰 수의 값을 작은 수로 나눈다. 그리고 남은 나머지로 작은 수를 또 나눠준다. 이런 식으로 반복하면 결국 나머지가 0 이 되는 순간이 온다. 그 순간의 작은 수가 바로 최소공배수가 된다. 최대공약수는 더 간단한데, 두 수를 곱하고 최소배수로 나누면 최대공약수가 된다. 두 수를 곱하는 과정에서 나올 수 있는 overflow 는 long long int 형을 사용하면 쉽게 해결할 수 있다.

서식 지정함: 글꼴: 10 pt

## ❑ 고찰

❑ 결국 최소공배수를 구할 때 두 수의 곱을 이용하는데, 입력된 두 수의 곱이 int형의 범위를 넘어가면 overflow가 발생한다. 그래서 long long int형을 선언해주면 일반적인 int형보다 범위가 훨씬 크기 때문에 overflow 문제를 해결할 수 있다. 그러나 long int 또한 32비트라고 하는데 int의 경우에도 32비트다. 교수님께서 같은 비트로 더 넓은 범위를 표현한다는 것은 그만큼 오차가 생길 확률이 넓은 것과도 같다고 하셨는데 long int와 int의 경우에도 똑같이 적용되는지 궁금하다.

## Program 4

### ❑ 문제 설명

- ❑ **3x3행렬을 만들고 해당 행렬을 통해 다양한 계산을 해라.**
- ❑ 3X3행렬에서 cofactor 값만 제대로 구한다면 나머지 값들은 쉽게 구할 수 있다. 결국 1번 문제와 같은 아이디어로 2차원 배열을 좌표라고 생각하고 접근하면 쉽게 해결할 수 있다.

## ❑ 결과 화면

Microsoft Visual Studio 디버그 콘솔

```
input a~i : 1
0
2
2
-1
0
1
1
1
Get and check determinant of the matrix
det(A) = 5

Get the cofactor matrix C
-1   -2   3
2   -1   -1
2   4   -1

Transpose the cofactor matrix C^T
-1   2   2
-2   -1   4
3   -1   -1

Divide each elements using determinant A^-1
-0.2   0.4   0.4
-0.4   -0.2   0.8
0.6   -0.2   -0.2
```

단순하다. Cofactor 값만 구한다면 나머지 값은 어렵지 않다. Det (A)는 cofactor 의 배열 값을 모두 더하면 쉽게 구할 수 있고 transpose 의 경우 cofactor 를 담은 배열의 행과 열을 뒤집어서 출력해주면 구할 수 있다. 그리고 A^-1 의 경우에는 Transpose 를 5 로 나눠서 출력해주면 쉽게 출력할 수 있다.

서식 지정함: 글꼴: 10 pt

## ❑ 고찰

❑ 문제 자체는 어려운 문제가 아니었다. 2차원 배열을 통해서 함수를 행렬을 만들고 배열 하나 하나를 좌표로 생각하여 수학을 계산할 때 좌표를 이용하는 것처럼 계산했다. 그러나 아쉬운 것은 cofactor matrix를 계산할 때 단순노동이 상당히 많았는데, 이러한 부분을 단순노동이 아닌 패턴을 만들어서 반복문으로 돌릴 수는 없었을까 하는 생각이 들었다. 분명 규칙성이 있을 텐데 발견하지 못한 아쉬움이 있다.

서식 지정함