

## CSED 232 Object-Oriented Programming (Spring 2022)

### Programming Assignment # 4

#### - Templates & STL -

Due date : 5월 20일 23시 59분

담당 조교 : 김건웅 (k2woong92@postech.ac.kr)

#### 주의 사항

- 클래스 선언 및 정의를 `main.cpp` 파일에 작성하는 것을 금지합니다. 그 외에 선언 및 정의의 위치에 대한 제약은 없습니다.
- STL 사용이 가능합니다.
- 모든 C++ 문법이 사용 가능합니다.
- 문제에서 제공한 형식을 준수해야 합니다.
- 문제에 명시되어 있지 않더라도 소멸자(Destructor)와 같은 메모리 누수 방지를 위해 필요한 멤버함수는 필수적으로 구현되어야 합니다.
- 각 문제별 추가적인 세부 조건을 만족하여야 합니다.
- 문제 조건이 복잡합니다. 모든 문제의 세부 조건을 꼼꼼히 읽어 보시기 바랍니다.

#### 감점

- 제출 기한에서 하루(24시간) 늦을 때마다 20%씩 감점
  - 1일(20%) , 2일(40%), ... 5일(100%)
- 컴파일이 정상적으로 이루어지지 않을 경우 0점

#### 제출방식

채점은 Windows Visual Studio 2022(윈도우 사용자의 경우) 및 Ubuntu 20.04(lts)와 gcc version 9.4.0 (맥 사용자의 경우) 환경에서 이루어집니다. VS로 작업했을 경우 작업하신 환경이 있는 visual studio 프로젝트 폴더에 Report 를 포함하여 zip 파일로 압축 후 제출해 주시기 바랍니다. (x64 및 .vs 폴더는 전부 지워주십시오) 마찬가지로, 맥 이용자의 경우 소스 코드, 보고서, Makefile 을 포함한 폴더를 압축해서 제출해주시면 됩니다. 폴더명은 '학번'으로 만들어 주시고, Report 는 docx 나 pdf 형식으로 제출해주세요. 반드시 PLMS 를 통해 제출해주시기 바랍니다. 이메일 제출은 인정되지 않습니다.

## 공통 채점 기준

### 1. 프로그램 기능

- 프로그램이 요구 사항을 모두 만족하면서 올바로 실행되는가?

### 2. 프로그램 설계 및 구현

- 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
- 문제에서 제시된 세부 조건을 모두 만족하였는가?
- 설계된 내용이 요구된 언어를 이용하여 적절히 구현되었는가?

### 3. 프로그램 가독성

- 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
- 변수 명이 무엇을 의미하는지 이해하기 쉬운가?
- 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

### 4. 보고서 구성 및 내용, 양식

- 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
- 보고서의 양식을 잘 따랐는가?

다른 사람의 프로그램이나 인터넷에 있는 프로그램을 복사(copy)하거나 간단히 수정해서 제출하면 학점은 무조건 'F'가 됩니다. 이러한 부정행위가 발견되면 학과에서 정한 기준에 따라 추가적인 불이익이 있을 수 있습니다.

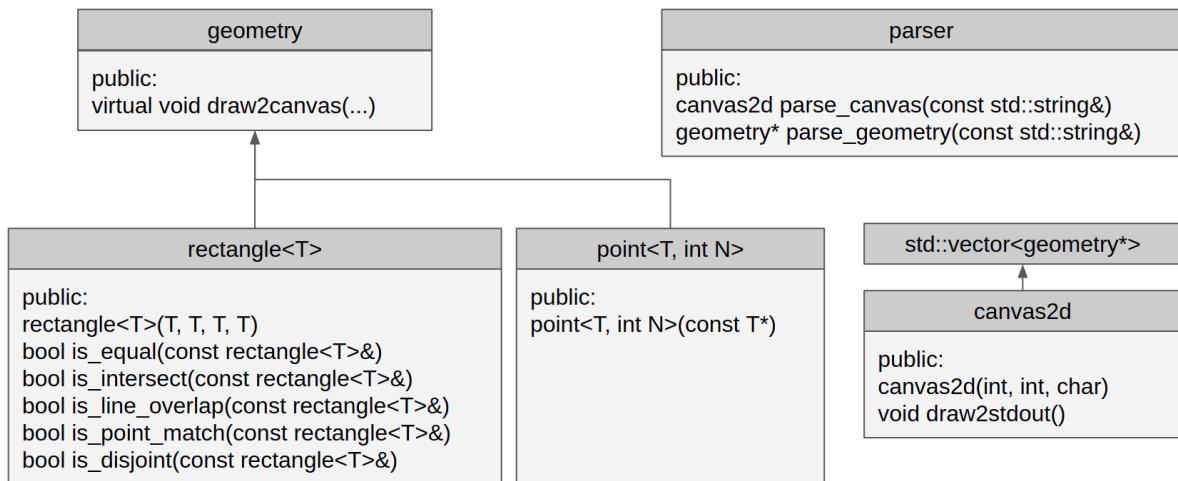
## SimpleGeoWorld

본 과제 “SimpleGeoWorld”는 기하(geometry) 객체들을 정의하고 이와 관련된 간단한 기능들을 구현해보면서 **template** 문법 사용에 친숙해지는 것을 목적으로 한다.

해결할 문제는 크게 두 가지다.

1. point와 rectangle를 standard output(이하 `stdout`)으로 표현
2. 임의의 두 rectangle 사이의 관계를 추정하는 함수 작성

문제 해결을 위해 사용되는 **class**와 **class** 사이의 상속 관계는 아래 그림 1과 같다. 화살표는 상속을 의미하고 방향표시가 있는 **class**가 부모 **class**다. “`<...>`”는 **template**으로 정의된 **class**를 의미한다. 함수의 인자가 “`(...)`”으로 표기된 것은 코드 작성자가 필요한 형식으로 정의 할 수 있음을 의미한다.



[ 그림1. Class diagram ]

프로그램을 구현에 아래와 같은 제약사항들이 따른다.

- 그림1에 명시된 멤버변수 및 함수는 반드시 구현되고 사용되어야 한다.
- 그림1에 명시된 멤버함수의 인자와 **return type**이 준수되어야 한다.
- 그림1에 명시된 **class**외에 다른 **class**를 정의해서 사용할 수 없다.
- 제공된 파일에 정의된 함수(function)외에 함수를 정의해서 사용할 수 없다.

- class diagram에 명시된 멤버함수(method)와 멤버변수 외에도 필요한 혹은 필수적인 멤버변수 및 멤버함수를 자유롭게 추가할 수 있다.

그림 1에 표현된 각 클래스에 대한 자세한 설명은 아래와 같다.

### **geometry**

- 모든 geometry 구성 요소들의 abstract class
- void draw2canvas(...)

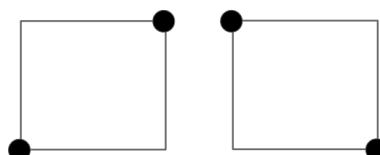
  - canvas의 instance가 geometry를 그릴 때 사용된다.
  - 인자(argument)형식은 문제작성자가 필요한 대로 정의한다.

### **point<T, int N>**

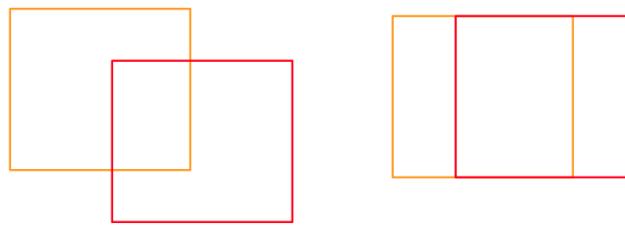
- 포인트 class로 T는 사용될 numeric type, N은 coordinate dimension을 의미한다
  - 예로, N이 2면 2차원 point, 3이면 3차원 point가 된다.
- point<T, int N>(const T\* coords)
  - point class의 생성자이며 N개의 T값으로 초기화 된다.

### **rectangle<T>**

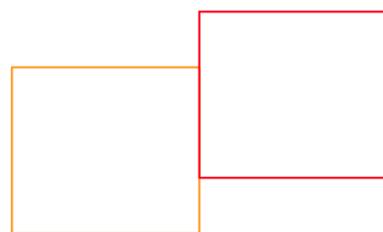
- 2차원 평면을 가정한 사각형 클래스
- 직각사각형, 각 변이 x,y축과 평행을 가정한다.
- rectangle<T>(T, T, T, T)
  - 아래 그림과 같이 사각형은 두 점의 좌표로 정의될 수 있다. 각각의 인자는 순서대로  $x_1, y_1, x_2, y_2$ 에 해당한다.



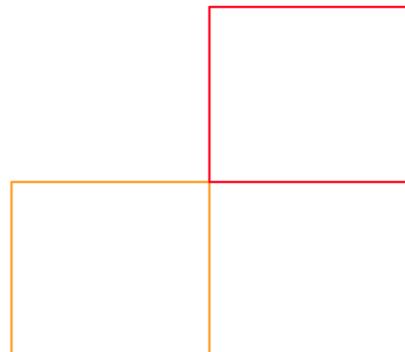
- 사각형의 면적이 0이 되도록 인자가 주어진 경우 “throw” 키워드를 통해서 프로그램을 fail시킨다.
- bool is\_equal(const rectangle<T>& other)
  - 멤버함수를 호출하는 instance와 other이 정확히 4개의 point를 공유하는 경우 true, 그렇지 않은 경우 false
- bool is\_intersect(const rectangle<T>& other)
  - 멤버함수를 호출하는 instance와 other이 equal이 아니면서 영역이 겹치는 경우 true, 그렇지 않은 경우 false (아래 예시)



- `bool is_line_overlap(const rectangle<T>& other)`
  - 멤버함수를 호출하는 `instance`와 `other`이 `equal` 및 `intersect`가 아니면서 한 변을 공유하는 경우 `true`, 그렇지 않은 경우 `false`(아래 예시)



- `bool is_point_match(const rectangle<T>& other)`
  - 멤버함수를 호출하는 `instance`와 `other`이 `intersect`가 아니면서 오직 한 점을 공유하는 경우 `true`, 그렇지 않은 경우 `false` (아래 예시)



- `bool is_disjoint(const rectangle<T>& other)`
  - 멤버함수를 호출하는 `instance`와 `other`이 만나지 않는 경우 `true`, 그렇지 않은 경우 `false` (아래 예시)



## canvas2d

- STL vector를 상속 받아 구현되며 보유한 객체를 stdout으로 그려내는 class
- canvas2d(int width, int height, char c\_empty)
  - width: canvas의 폭
  - height: canvas의 높이
  - 빈 공간을 표현할 글자(char)
- void draw2stdout()
  - 보유한 객체를 stdout으로 출력

```
canvas2d c(5, 3, '.');
c.draw2stdout();
```



```
canvas2d c(3, 5, '@');
c.draw2stdout();
```

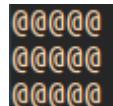


[ code와 stdout 예시 ]

## parser

- string을 받아 객체의 instance를 생성하는 기능을 제공하는 class
  - canvas2d parse\_canvas(const std::string& line)

```
parser p;
canvas2d c = p.parse_canvas("5,3,@");
c.draw2stdout();
```



[ code와 stdout 예시 ]

- geometry\* parse\_geometry(const std::string& line)

```
parser p;
canvas2d c = p.parse_canvas("5,3,.");
geometry* g = p.parse_geometry("point,int,1,2");
c.push_back(g);
c.draw2stdout();
```



[ code와 stdout 예시 ]

본격적인 문제 설명에 앞서서 아래와 같은 조건 및 가정이 있다.

- template에 사용되는 type은 int와 float으로 가정한다.
- 문제1과 문제2는 채점은 stdin과 stdout으로 진행된다. (input 및 output파일을 사용하지 않는다)
- 문제3은 레포트에 작성한다.
- 아래 주어진 **int main()** 함수와 **void print\_relation(rectangle<T>, rectangle<T>)** 함수는 절대로 수정하면 안된다.

```

template<typename T>
void print_relation(rectangle<T> r1, rectangle<T> r2)
{
    bool a1 = r1.is_equal(r2);
    bool a2 = r1.is_intersect(r2);
    bool a3 = r1.is_line_overlap(r2);
    bool a4 = r1.is_point_match(r2);
    bool a5 = r1.is_disjoint(r2);

    std::cout
        << a1 << ","
        << a2 << ","
        << a3 << ","
        << a4 << ","
        << a5 << std::endl;
}

int main()
{
    std::string p_type;
    std::cin >> p_type;

    // Solve problem 1
    if (p_type == "problem1")
    {
        std::string line;
        parser parser;
        std::cin >> line;
        canvas2d canvas = parser.parse_canvas(line);

        while (std::cin >> line) {
            geometry* geo = parser.parse_geometry(line);
            canvas.push_back(geo);
        }
        canvas.draw2stdout();
    }

    // Solve problem 2
    else if (p_type == "problem2")

```

```
{  
    std::string line;  
    while (std::cin >> line)  
    {  
        std::stringstream ss(line);  
        std::string type;  
        std::getline(ss, type, ',');  
  
        if (type == "int")  
        {  
            std::vector<int> vals;  
            while (ss.good())  
            {  
                getline(ss, type, ',');  
                vals.push_back(std::stoi(type));  
            }  
            rectangle<int> rec1(vals[0], vals[1], vals[2], vals[3]);  
            rectangle<int> rec2(vals[4], vals[5], vals[6], vals[7]);  
            print_relation<int>(rec1, rec2);  
        }  
        else if (type == "float")  
        {  
            std::vector<float> vals;  
            while (ss.good())  
            {  
                getline(ss, type, ',');  
                vals.push_back(std::stof(type));  
            }  
            rectangle<float> rec1(vals[0], vals[1], vals[2], vals[3]);  
            rectangle<float> rec2(vals[4], vals[5], vals[6], vals[7]);  
            print_relation<float>(rec1, rec2);  
        }  
        else throw;  
    }  
}  
else throw;  
return 0;  
}
```

**문제1> canvas2d instance의 draw2stdout() 함수를 사용하여 geometry instance들을 적절하게 그리시오.**

그림1처럼, canvas2d class는 std::vector<geometry\*>를 상속받아 구현되며 geometry\* instance들을 갖는 container로 볼 수 있다. 해당 클래스의 canvas2d instance는 draw2stdout() 함수를 호출해서 자신이 갖는 geometry instance들을 stdout으로 그려낸다.

문제의 입출력(stdin, stdout)에 대한 설명은 아래와 같다.

입력 line 1

- 풀고자 하는 문제를 “problem1”으로 명시한다. 해당 과제는 두 문제를 제시하고 있기 때문에 문제 사이의 구분이 필요하다.

입력 line 2

- 초기화된 canvas2d 인스턴스의 인자를 의미한다. ','로 구분된 첫 번째 값은 canvas의 폭(width), 두 번째 값은 높이(height), 세 번째 값은 빈공간을 표시할 글자(char)를 의미한다.
- canvas2d(width, height, c\_empty)

입력 line 3 ~ <EOF>

- 세 번째 줄부터는 생성될 geometry instance의 명세가 표시된다. 첫 번째 값은 class 이름, 두 번째 값은 template type, 그 이후 값은 생성자 호출에 사용되는 인자를 의미한다.

problem1 5,4,. point,int,2,3 point,int,0,0	
---	---

[ stdin | stdout 예시 ]

각각의 geometry에 대한 표현 방법은 아래와 같다.

- 2차원 point는 해당되는 위치에 숫자 1이 표시된다. 여기서 해당되는 위치는 첫 번째 차원의 숫자를 x좌표 두 번째 차원의 숫자를 y좌표로 보았을 때 위치다. 또한 x 축은 원쪽에서 오른쪽, y축은 위에서 아래 방향으로 가정한다.

problem1 5,4,. point,int,2,3	
------------------------------------	---

[ point stdin | stdout 예시 ]

- 3차원 point 는 해당되는 위치에 세 번째 차원 값이 표시된다.

```
problem1
5,4,.
point,int,2,3,6
```



[ point stdin | stdout 예시 ]

- 4차원 이상의 point 는 해당되는 위치에 세 번째 차원부터 끝 차원 값을 모두 더한 값이 표시된다.

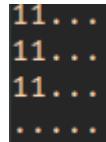
```
problem1
5,4,.
point,int,2,3,6,2
```



[ point stdin | stdout 예시 ]

- rectangle은 해당되는 영역에 숫자 1이 표시된다.

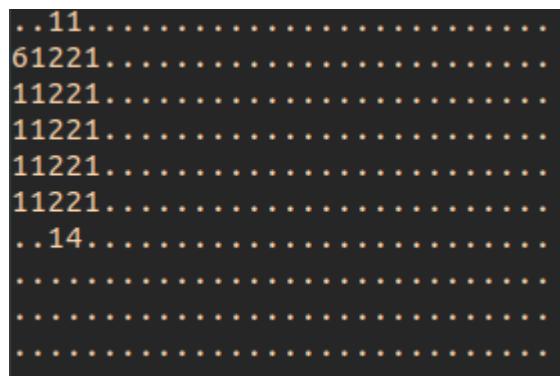
```
problem1
5,4,.
rectangle,int,0,0,1,2
```



[ rectangle stdin | stdout 예시 ]

- canvas에 영역이 겹쳐서 그려지는 경우 아래와 같이 더해진(+) 숫자가 표시된다.

```
problem1
30,10,.
point,int,0,1
point,float,3.33,6.23
point,int,0,1,4
point,float,3.33,6.23,2.22
rectangle,int,0,1,4,5
rectangle,float,3.33,6.23,2.22,0.0
```



[ overlap case stdin | stdout 예시 ]

좌표계는 위 예시처럼 좌측 상단이 (0,0)이고 x축과 y축은 각각 가로 세로축이다. 추가적으로 아래와 같은 조건이 있다.

- numeric type T, 즉 위에서 사용되는 datatype T는 float 또는 int로 가정한다.
- float type의 좌표를 stdout으로 그리는 경우 소수점 이하 값을 버려서 사용한다.
  - 그릴때 마다 소수점 이하를 버리야 합니다. 예를 들어서  
 point,float,2.0,3.0,1.3  
 point,float,2.1,3.7,1.7  
 위와 같은 경우 (2,3)에 3이 아닌 2가 표기되어야 합니다.
- geometry 객체가 캔버스 범위를 벗어나는 경우 무시한다.
- 숫자가 10 9을 넘어가면 다시 0부터 시작된다.
  - 해당 조건이 의도한 바는 0-9까지의 숫자만을 사용하는 것입니다. 예를 들어, 합이 10이면 0을 출력해야합니다.
- point의 차원은 2이상 5 이하로 가정한다.

해당 문제에 세부 조항은 아래와 같다

- stdin으로 받은 string에서 적절한 instance를 생성할 수 있도록 parser 멤버함수 구현
- 각각의 geometry instance들이 자기 자신에 적절한 그림을 그리도록 구현
- canvas instance가 적절한 stdout을 출력하도록 멤버함수 구현
- 다양한 stdin에 대해서 정확하게 동작하도록 구현

## 문제2> 사각형 사이의 관계정의

위 클래스 설명에서 기술한 바와같이 두 rectangle 사이에 equal, intersect, line\_overlap, point\_match, disjoint 라는 다섯가지 관계를 설명했다. 여기서 두 **rectangle** 사이의 관계는 상호 배타적(**mutual exclusive**)이다. 즉, 임의의 두 rectangle instance는 오직 하나의 관계로 정의되어야 한다. 예를 들어서 equal과 동시에 intersect일 수 없다. 해당 조건을 만족하도록, 아래 다섯가지 함수를 정확하게 구현해야한다.

- bool rectangle<T>::is\_equal(const rectangle<T>& other)
- bool rectangle<T>::is\_intersect(const rectangle<T>& other)
- bool rectangle<T>::is\_line\_overlap(const rectangle<T>& other)
- bool rectangle<T>::is\_point\_match(const rectangle<T>& other)
- bool rectangle<T>::is\_disjoint(const rectangle<T>& other)

문제의 입출력에 대한 설명은 아래와 같다.

입력 line 1

- 풀고자하는 문제를 “problem2”으로 명시한다.

입력 line 2~ <EOF>

- ','로 구분된 첫 번째 값은 rectangle instance 생성에 사용될 datatype이고 이어서 나오는 8개의 숫자는 2개의 rectangle instance 생성에 사용될 값이다. 입력에 대한 rectangle instance 생성 예시는 아래와 같다.
  - int, rec1\_x1, rec1\_y1, rec1\_x2, rec1\_y2, rec2\_x1, rec2\_y1, rec2\_x2, rec2\_y2
    - rectangle<int> rec1(rec1\_x1, rec1\_y1, rec1\_x2, rec1\_y2)
    - rectangle<int> rec2(rec2\_x1, rec2\_y1, rec2\_x2, rec2\_y2)

problem2

int,0,1,2,3,4,5,6,7

float,0.3,1.2,4.3,8.4,2.3,8.4,8.2,8.3

[ stdin 예시 ]

출력 line 1~

- “equal”, “intersect”, “line\_overlap”, “point\_match”, “disjoint”에 대한 bool 결과 값을 출력한다.
- 두 개 이상의 항목에서 true(1)이 나오면 mutual exclusive가 아니므로 오답이다.

0,0,0,0,1

0,1,0,0,0

[ stdout 예시 ]

문제3> canvas2d class 처럼 STL vector를 상속받을때 예상되는 단점을 간단하게 논하시오.

## Appendix

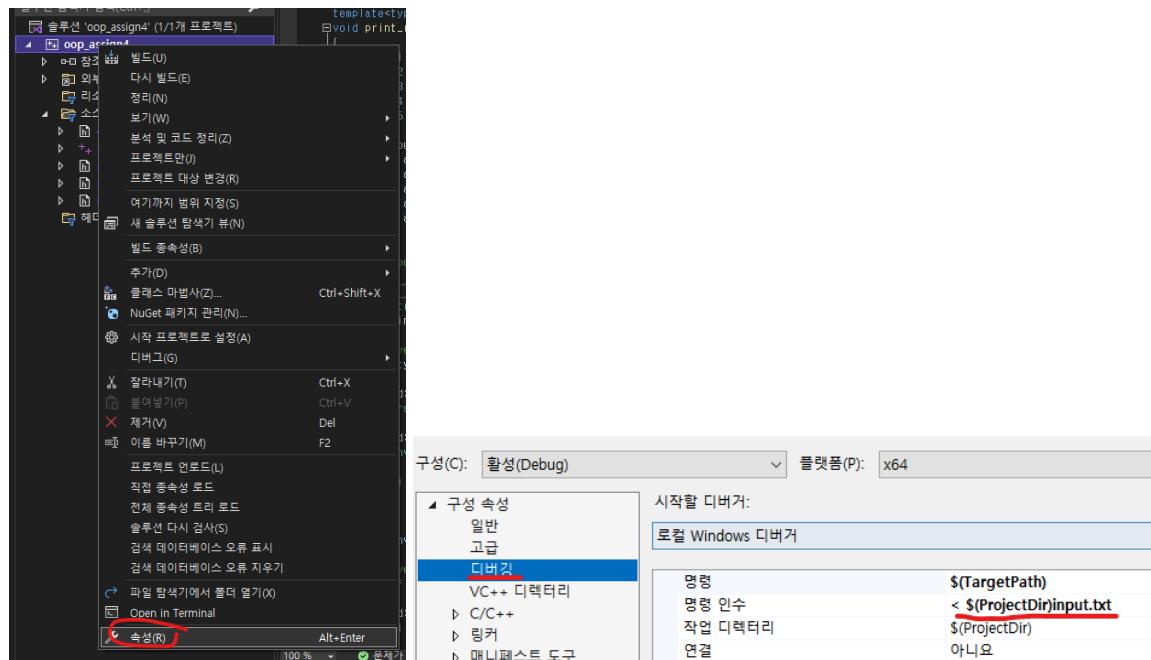
Appendix에는 과제 수행에 있어서 필요한 부가적인 설명이나, 빈도높은 질문에 대한 대답을 작성한 단락입니다.

### <EOF> End Of File

- 과제와 같이 `stdin`을 입력으로 받는 프로그램은 많은 경우 EOF라는 `signal`을 통해 입력을 끝내게 됩니다. EOF를 보내는 key는 os마다 다르고 **window**의 경우 **ctrl-z**, **linux**는 **ctrl-d**를 사용합니다. (여기서 **window**의 경우 **ctrl-z** 누른 후 **enter** 까지 눌러야 동작하는 경우가 있습니다.)

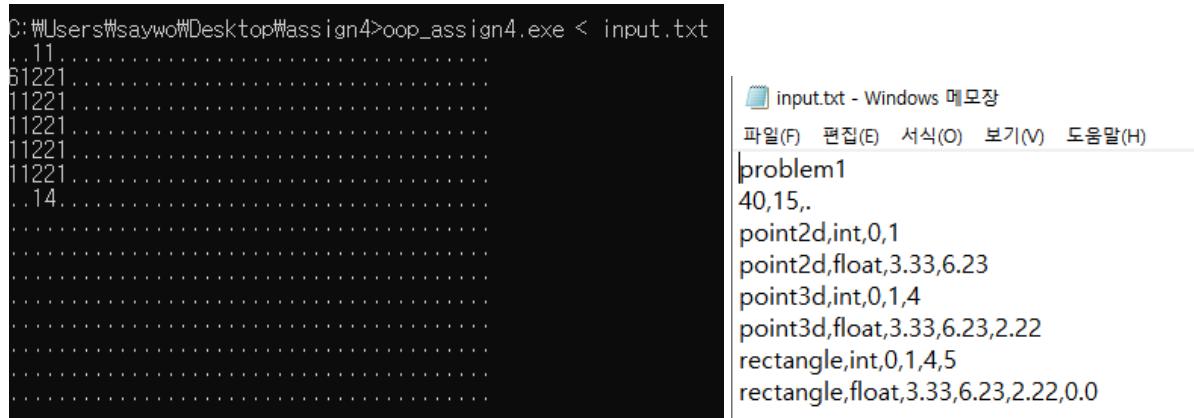
### Visual studio `stdin` 디버그

- 파일로부터 `string`을 불러오지 않기 때문에 편리한 디버깅을 위해 아래와 같은 설정을 하면 유용합니다. 먼저 **project** 폴더에 원하는 `stdin` `string`이 적힌 파일을 생성합니다. 아래 예시에서는 “`input.txt`”로 되어있습니다. 여기서 **project** 폴더는 `*.vcxproj` 파일이 있는 위치를 의미합니다. 그 후 속성 → 디버깅 → 명령인수에 “`< $(ProjectDir){파일이름}`”을 설정하면 디버깅 모드에서 `stdin`으로 `string`을 불러옵니다.



## 파일 실행

- **Stdin**으로 입력받아 동작하는 프로그램은 타이핑하는 것이 번거로울 수 있습니다. 이럴때는 **stdin**으로 입력받을 내용을 **text**파일로 저장시켜두고 아래와 같이 **cmd**에서 동작시키면 **text** 파일의 내용이 **stdin**으로 들어가 동작하게 됩니다. (이것은 리눅에서도 동일하게 동작합니다)



The screenshot shows a terminal window on the left and a Windows Notepad window on the right. The terminal window displays command-line input and output. The Notepad window shows a text file named 'input.txt' containing several lines of data.

Terminal Output (C:\Users\saywo\Desktop\assign4>oop\_assign4.exe < input.txt):

```
11  
11221  
11221  
11221  
11221  
11221  
14
```

Notepad Content (input.txt - Windows 메모장):

```
input.txt - Windows 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
problem1  
40,15.,  
point2d,int,0,1  
point2d,float,3.33,6.23  
point3d,int,0,1,4  
point3d,float,3.33,6.23,2.22  
rectangle,int,0,1,4,5  
rectangle,float,3.33,6.23,2.22,0.0
```