R E P O R T

실습과제 09

내용

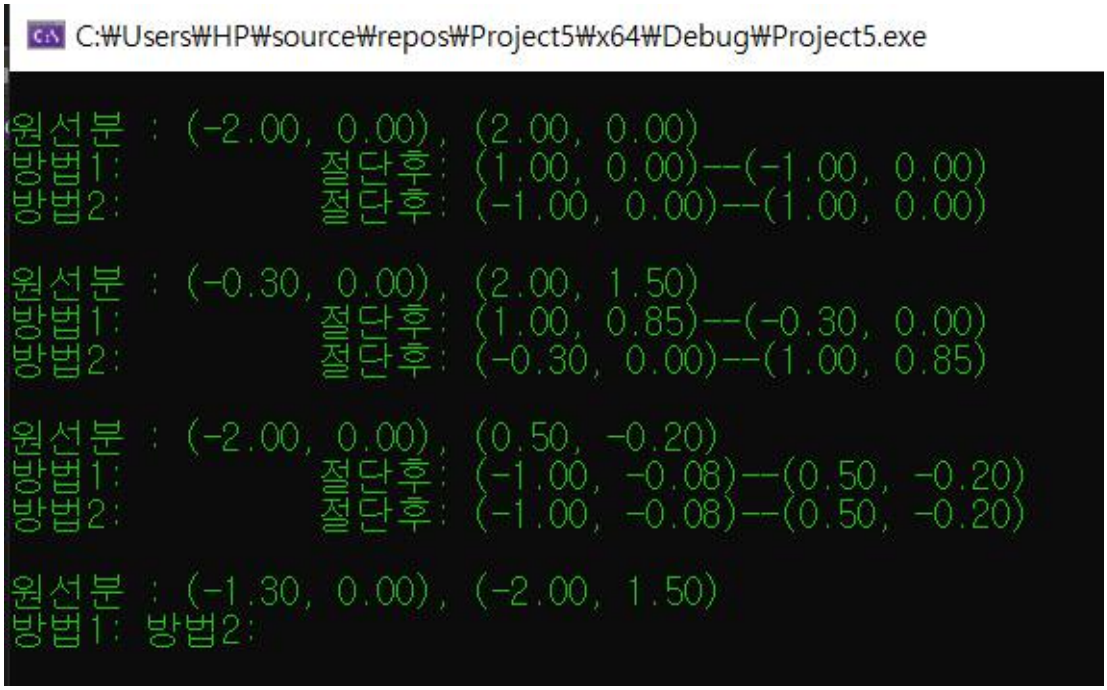
과제 명세
선분 절단 알고리즘 구현하기
고찰 및 느낀점

과제 명세

1. 선분 절단 알고리즘 구현하기
   - 코헨-서덜랜드의 알고리즘 구현: clipLine_CS()
   - 리앙-바스키 알고리즘 구현: clipLine_LB()
   - 테스트 코드 작성
     - 다음 코드 참조
     - 콘솔 응용 프로그램으로 작성해도 됨
     - 여러 영역의 점들을 이용해 테스트할 것

선분 절단 알고리즘 구현하기

- 결과물



- 코헨-서덜랜드 알고리즘 구현하기

```
// 코헨-서덜랜드 알고리즘

// LEFT, RIGHT, BOTTOM, TOP을 각각 0001 0010 0100 1000을 이용한
// 비트 연산으로 읽을 수 있도록 다음과 같이 Encode 메서드를 정의했다.
enum Boundary { LEFT = 0x01, RIGHT = 0x02, BOTTOM = 0x04, TOP = 0x08};
char Encode(Point2D p, Point2D min, Point2D max) {
	unsigned char code = 0x00;
	if (p.x < min.x) code |= LEFT;
	if (p.x > max.x) code |= RIGHT;
	if (p.y < min.y) code |= BOTTOM;
	if (p.y > max.y) code |= TOP;
	return (code);
}

// Swap_If_Nedded x1, y1이 외부에 있도록 바꿔주는 함수
// p1과 p2에 동일한 결과를 이끌어내기 위해 다음과 같은 과정을 진행했다.
void Swap_If_Nedded(Point2D& p1, Point2D& p2, char& c1, char& c2) {
	if (c1 == 0) {
		Point2D r = p1;
		p1 = p2;
		p2 = r;

		char tmp = c1;
		c1 = c2;
		c2 = tmp;
	}
}

// Accept 조건 : (c1 OR c2) == 0
bool Accept(char c1, char c2) {
	return (c1 | c2) == 0;
}
// Reject 조건 : (c1 AND c2) ≠ 0
bool Reject(char c1, char c2) {
	return (c1 & c2) != 0;
}
// drawLine : 코헨-서덜랜드와 리앙-바스키 알고리즘의 결과를 출력하는 메서드이다.
void drawLine(Point2D p1, Point2D p2) {
	printf("₩t₩t절단후: (%4.2lf, %4.2lf)--(%4.2lf, %4.2lf)₩n", p1.x, p1.y, p2.x, p2.y);
}


// 코헨-서덜랜드 알고리즘 구현
void clipLine_CS(Point2D min, Point2D max, Point2D p1, Point2D p2) {
	char code1, code2;
	int done = false, display = false;
	int i = 0;
	while (!done) {
		code1 = Encode(p1, min, max);
```

```
                code2 = Encode(p2, min, max);
                //printf("code1 : %d₩tcode2 : %d₩n", code1, code2);
                if (Accept(code1, code2)) {
                        done = display = true;
                }
                else if (Reject(code1, code2)) {
                        done = display = false;
                        return;
                }
                else {
                        Swap_If_Nedded(p1, p2, code1, code2);           // 점의 위치를 바꿔줌.
                        double m = (p2.y - p1.y) / (p2.x - p1.x);  // 기울기
                        if (code1 & LEFT) {
                                p1.y = p1.y + (min.x - p1.x) * m;       // 수직 경계와의 교차
                                p1.x = min.x;
                                code1 -= LEFT;
                        }
                        else if (code1 & RIGHT) {
                                p1.y = p1.y + (max.x - p1.x) * m;       // 수직 경계와의 교차
                                p1.x = max.x;
                                code1 -= RIGHT;
                        }
                        else if (code1 & BOTTOM) {
                                p1.x = p1.x + (min.y - p1.y) / m;       // 수평 경계와의 교차
                                p1.y = min.y;
                                code1 -= BOTTOM;
                        }
                        else if (code1 & TOP) {
                                p1.x = p1.x + (max.y - p1.y) / m;       // 수평 경계와의 교차
                                p1.y = max.y;
                                code1 -= TOP;
                        }
                }
        }
        drawLine(p1, p2);
}
```

- **설명**

LEFT : 0001 RIGHT : 0010 BOTTOM : 0100 TOP : 1000

1001	1000	1010
0001	0000	0010
0101	0100	0110

위 같은 2차원 좌표를 기준으로 아래와 같은 형태로 이루어집니다.

[0,1]	[0, 1]	[0,1]	[0, 1]
TOP	BOTTOM	RIGHT	LEFT

비트 연산을 통해 두 점의 OR 연산이 0인 경우에는 ACCEPT를 해주고 AND 연산이 1인 경우에는 Reject를 해줍니다

그리고 수직 경계의 교차와 수평 경계의 교차를 기울기를 중심으로 계산을 해주어 다음과 같은 코드를 작성해줍니다.

```
else {
        Swap_If_Nedded(p1, p2, code1, code2);          // 점의 위치를 바꿔줌.
        double m = (p2.y - p1.y) / (p2.x - p1.x);  // 기울기
        if (code1 & LEFT) {
                p1.y = p1.y + (min.x - p1.x) * m;      // 수직 경계와의 교차
                p1.x = min.x;
                code1 -= LEFT;
        }
        else if (code1 & RIGHT) {
                p1.y = p1.y + (max.x - p1.x) * m;      // 수직 경계와의 교차
                p1.x = max.x;
                code1 -= RIGHT;
        }
        else if (code1 & BOTTOM) {
                p1.x = p1.x + (min.y - p1.y) / m;      // 수평 경계와의 교차
                p1.y = min.y;
                code1 -= BOTTOM;
        }
        else if (code1 & TOP) {
                p1.x = p1.x + (max.y - p1.y) / m;      // 수평 경계와의 교차
                p1.y = max.y;
                code1 -= TOP;
        }
}
```

y = y1 + m(x – x1) : 수직 경계와의 교차

x = x1 + (y – y1) / m : 수평 경계와의 교차

m : 기울기 : (y2 – y1) / (x2 – x1)

=> 이와 같은 과정을 통해 코헨-서덜랜드 알고리즘이 완성됩니다.

조건에 맞지 않는 경우는 출력하지 않도록 drawLine을 세팅해주었습니다.

- 리앙-바스키 알고리즘 구현하기

```
bool clipTest(double p, double q, double* u1, double* u2) {
	double r = 0.0;
	bool result = true;
	if (p < 0.0) {		// 외부에서 내부로 들어감
		r = (double)q / p;
		if (r > *u2) result = false;
		else if (r > *u1) {
			*u1 = r;
			return true;
		}
	}
	else if (p > 0.0 ) {	// 내부에서 외부로 나감
		r = (double) q / p;
		if (r < *u1) result = false;
		else if (r < *u2) {
			*u2 = r;
			return true;
		}
	}
	else {					// 수직 또는 수평선(평행선)
		if (q < 0) result = false;
	}
	return result;
}
void clipLine_LB(Point2D min, Point2D max, Point2D p1, Point2D p2) {
	double U1 = 0, U2 = 1;
	double *u1 = &U1, *u2 = &U2;
	double dx = p2.x - p1.x;
	double dy = p2.y - p1.y;
	// printf("dx, dy : %lf %lf₩n", dx, dy);
	if (clipTest(-dx, p1.x - min.x, u1, u2)) {                                  //left
		if (clipTest(dx, max.x - p1.x, u1, u2)) {                               //right
			if (clipTest(-dy, p1.y - min.y, u1, u2)) {                  //lower
				if (clipTest(dy, max.y - p1.y, u1, u2)) {           //upper
					if (*u2 < 1.0) {
						p2.x = p1.x + (*u2) * dx;
						p2.y = p1.y + (*u2) * dy;
					}
					if (*u1 > 0.0) {
						p1.x = p1.x + (*u1) * dx;
						p1.y = p1.y + (*u1) * dy;
					}
					drawLine(p1, p2);
				}
			}
		}
	}
}
```

- **설명**

코헨-서덜랜드의 경우 양 끝점을 기준으로 상대적 위치에 따라 비트연산을 통해 진행했다면 리앙-바스키 알고리즘의 경우에는 선분의 매개변수 방정식을 이용해서 P = 0, P > 0, P < 0일 때의 세 가지 경우를 가지고

상하좌우에 해당하는 네 개의 부등식을 정의한 후에

```
if (clipTest(-dx, p1.x - min.x, u1, u2)) {                              //left
                    if (clipTest(dx, max.x - p1.x, u1, u2)) {                       //right
                                if (clipTest(-dy, p1.y - min.y, u1, u2)) {          //lower
                                            if (clipTest(dy, max.y - p1.y, u1, u2)) {       //upper
```

• • •

평행선인 경우(P = 0), 선분이 외부에서 내부로 들어가는 경우(P < 0), 선분이 내부에서 외부로 나가는 경우(P > 0)에 따라 절단 알고리즘을 구현합니다.

코헨-서덜랜드와 다르게 점을 바꿔주는 과정이 없습니다.

- 테스트 코드 작성

```
void lineClipTest(Point2D min, Point2D max, Point2D p, Point2D q) {
	printf("₩n원선분 : (%4.2lf, %4.2lf), (%4.2lf, %4.2lf)₩n"
		, p.x, p.y, q.x, q.y);
	printf("방법1: ");
	clipLine_CS(min, max, p, q);
	printf("방법2: ");
	clipLine_LB(min, max, p, q);
}
int main(int argc, char** argv) {
	Point2D min = { -1, -1 };
	Point2D max = { 1,1 };
	Point2D p[8] = {
		{ -2, 0 }, { 2, 0 },
		{ -0.3, 0 }, { 2, 1.5 },
		{ -2, 0 }, { 0.5, -0.2 },
		{ -1.3, 0 }, { -2, 1.5 }
	};

	lineClipTest(min, max, p[0], p[1]);
	lineClipTest(min, max, p[2], p[3]);
	lineClipTest(min, max, p[4], p[5]);
	lineClipTest(min, max, p[6], p[7]);
	getchar();
	return 0;
}
```

고찰 및 느낀점

과제를 하면서 시간이 부족한 것을 느꼈다. 코드는 돌렸지만 의사 코드를 토대로 만들어서 아직은 공부가 더 필요한 것 같다. 점을 출력하는 형태로 구현을 진행했는데. 클리핑을 GUI상에서 구현하는 것은 어떻게 해야할지 조금 더 지켜봐야 될 것 같다.