문제해결 및 실습 : C++

과제 4,5 보고서

제출일	2024. 06.23	전공	소프트웨어학과
과목	문제해결및실습:C++	학번	21011746
담당교수	박상일 교수님	이름	양현석

1. MFC를 이용한 도형 그리기 프로그램 제작

a) 문제 분석

과제의 요구 조건들은 다음과 같습니다.

1) MyShape 기본 클래스 만들어서 이를 구체화시킨 도형 클래스에 상속

-> 간단히 상속을 통해 구현

2) 마우스 드래깅을 이용해서 도형 그리기(선택은 toolbar or menubar) 또한 순서가 유지되어야 함.

-> STL list나 vector로 구현 가능

3) 선택 기능을 통해서 각 도형을 선택할 수 있고, "Shift+왼쪽 클릭"이나 빈 곳에서 시작한 드래깅을 통해서 복수의 도형 선택 가능

-> 각각의 객체가 선택되었다는 것을 알려줄 각각의 flag가 필요함

4) 복수 혹은 단수의 도형을 선택 후 드래깅을 통해 위치를 변경할 수 있음.

-> 다 함께 움직일 수 있게끔 하고, 움직인 후에 그 도형의 모양을 유지해야 함.

5) 선택한 도형을 "Delete"키를 통해 삭제 가능

-> 간단하게 구현 가능

6) Grouping, UnGrouping을 할 수 있어야 함.

-> 트리구조를 활용해야 함. (실제로는 비슷하지만 다른 방법을 채택)

7) 순서를 바꿀 수 있어야 함. (맨 앞으로, 맨 뒤로)

-> 간단하게 구현 가능

8) 우클릭을 통해 Action 메뉴 호출

-> 간단하게 구현 가능

어떤 식으로 프로그램을 짜야 하는지 적어보았으니, 본격적으로 문제 풀이를 해보겠습니다.

b) 문제 풀이

가장 먼저 MyShape 클래스와 각 파생 클래스를 구현하겠습니다. MyShape 클래스는 기본 클래스로 써 도형의 기본적인 색상, 선택 여부, 선택 박스 요소(Border를 그리기 위함), 도형 타입 등과 나중에 그룹핑을 할 때 사용할 MyShape* 벡터를 만들어줍니다. 생성자는 기본 생성자, 도형 생성 전용 생성자(원, 박스), 그룹 객체 전용 생성자를 만듭니다. 멤버 함수로는 Border를 정해줄 setSelBox(), 그루핑을 도와줄 setGroup(), 그리고 도형들을 움직일 vitual 함수 move()를 생성합니다. 간단하게 멤버 변수만 보이겠습니다.

```
class MyShape {
public:
	//		BORDER 박스 요소
	CRect box;
	CPoint invb, bor;

	//		선택 여부
	bool isSel;

	//		색상
	COLORREF c;

	//		도형 타입
	// type -> 0 : RECT, 1 : CIRCLE, 2 : CURVE, 3 : STAR, 5 : SELECT
	int type;

	//		선택박스 요소
	CRect selBox;
	CPoint topLeft;
	CPoint bottomRight;

	//		그룹 요소
	vector<MyShape*> group;
	bool isG;

	MyShape* parent;
};
```

가장 중요한 점은 과연 멤버 변수인 MyShape* 벡터로 어떻게 그룹화를 하느냐 입니다. 트리에 담는 것이 가장 좋을 것이라고 문제에서 제시를 해주었습니다만, 기본 클래스의 포인터의 배열을 그룹으로 사용하면, 업캐스팅한 각 도형과 그룹 자체를 모두 MyShape*로 저장하여 가장 메인이 되는 shapeList에 담을 수 있다는 점이 좋을 것 같다고 생각했습니다. 이를 통해서 단복수의 그룹 혹은 단복수의 도형들을 자유롭게 배열에 담고 또 그룹을 해제할 때에도 그룹화된 그룹이 별도의 처리 없이 안전하게 나올 수 있었습니다. 물론 그 내부를 보기 위해서 트리와 마찬가지로 재귀함

수를 구현하고 사용하는 것이 상당히 까다로웠습니다.

다음은 사각형, 원, 곡선, 별을 생성해보겠습니다. 나머지는 사실 상 구현하는데 크게 어렵지 않기 때문에 별에 대해서만 설명하겠습니다. 별은 외접하는 원을 기준으로 구현합니다. 처음 클릭한 지점에서부터 드래그한 지점까지의 거리가 반지름, 그리고 내부의 점들은 0.4를 곱한 비율을 사용했습니다. 이렇게 구한 2개의 반지름으로 별들의 포인트를 지정합니다.

```
void setPoint() {
		for (int i = 0; i < 10; i++) {
			if (i % 2 == 1) {
				float x = (radius)*cos(i * PI * 72 / 360.f + PI * 36 / 360.f) + center.x;
				float y = (radius)*sin(i * PI * 72 / 360.f + PI * 36 / 360.f) + center.y;
				starList[i] = CPoint(x, y);
			}

			else {	//inner
				float x = (radius2)*cos(i * PI * 72 / 360.f + PI * 36 / 360.f) +
center.x;
				float y = (radius2)*sin(i * PI * 72 / 360.f + PI * 36 / 360.f) + center.y;
				starList[i] = CPoint(x, y);
			}
		}
	}
```

점은 총 10개, 즉 2*PI를 10으로 나눈 값을 이용하고, 화면의 정 중앙이 (0,0)이 아니기에 좌측 상단을 (0,0)으로 두게 끔 만들어줍니다. 그 후에 각 점들을 PolyRgn함수를 이용해서 별로 출력합니다. (CRgn이 아닌 PolyRgn을 통해 출력하는 이유는 경계선을 색칠하기 위해서 입니다)

이번에는 이동에 관련한 함수입니다. MyShape의 move()함수를 오버라이딩하여 사용합니다. 곡선과 별은 각각의 점들을 모두 이동하고 다시 그리지만, 원과 사각형은 그 도형의 CRect를 이용해서 옮기고 다시 그립니다. 아래는 곡선을 옮기는 Method입니다.

```
void move(CPoint vec) {
		vector<CPoint>::iterator it;

		for (it = curve.begin(); it != curve.end(); it++) {
			(*it) += vec;
		}

		invb += vec;
		bor += vec;
		box.SetRect(invb, bor);
```

```
                setBox();
                //          선택 박스 생성
                setSelBox();
        }
```

클래스에서 각각의 중요한 부분을 알아보았습니다. 그러면 이번에는 이 도형들을 어떻게 저장하느냐에 대한 알고리즘입니다. 모든 도형은 MyShape를 파생받아 사용하기에 업캐스팅으로 MyShape*를 저장하는 방식을 택했습니다. 후에 그림을 그릴 때는 필요에 따라서 다운캐스팅도 진행합니다. 안정성이 보장되지 않는 방법이지만, 가장 직관적이고 많은 테스트 후에 데이터 손실이 발생하지 않도록 구상하였습니다. 메뉴나 툴바를 이용해서 도형을 그릴 Method를 정하면 그에 따라 도형을 buff에 저장하고 그린다음, 그 buff를 배열에 저장합니다. 코드가 길기 때문에 저장하는 부분만 보이겠습니다.

```
                if (type == 0) {
                        if (p != CPoint(0, 0) && q != CPoint(0, 0)) {
                                shapeList.push_back(new MyRect(p, q, color));
                        }
                        p = CPoint(0, 0);
                        q = CPoint(0, 0);
                }

                else if (type == 1) {
                        if (buffCircle.click != CPoint(0, 0) && buffCircle.center != CPoint(0, 0)) {
                                shapeList.push_back(new MyCircle(buffCircle.center, buffCircle.click, color));
                        }

                        buffCircle.reset();
                }

                else if (type == 2) {
                        shapeList.push_back(new MyCurve(curve_point, color));

                        //          재 초기화
                        curve_point.clear();
                }

                else if (type == 3) {
                        if (buffStar.click != CPoint(0, 0) && buffStar.center != CPoint(0, 0)) {
                                shapeList.push_back(new MyStar(buffStar.center, buffStar.click, color));
                        }

                        buffStar.reset();
                }
```

도형을 저장했으면 다음은 그려야 할 차례입니다. 순서대로 그려야 하기 때문에 배열의 앞에서부터 뽑아주면서 진행힙니다.(Queue 알고리즘) 이때 끊김을 최소화하기 위해서 Double-Buffering

과 OnEraseBkgnd()를 사용합니다. 다음은 OnPaint에서 memDc에 그리는 코드입니다.

```
list<MyShape*>::iterator it;

	for (it = shapeList.begin(); it != shapeList.end(); it++) {
		if ((*it)->isG) {
			printAll((*it), &memDc);
		}
		CBrush brush((*it)->c);
		memDc.SelectObject(&brush);
		CPen black(PS_SOLID, 3, RGB(0, 0, 0));
		memDc.SelectObject(&black);

		if ((*it)->type == 0) {
			memDc.Rectangle((*it)->box);
		}

		else if ((*it)->type == 1) {
			memDc.Ellipse((*it)->box);
		}

		else if ((*it)->type == 2) {		//		curve

			//		다운 캐스팅
			CPen pen(BS_SOLID, 3, (*it)->c);
			memDc.SelectObject(pen);

			vector<CPoint>::iterator cp;
			MyCurve* tmp = (MyCurve*)(*it);
			if (tmp->curve.size() != 0) {
				for (cp = tmp->curve.begin(); cp != tmp->curve.end() - 1; cp++) {
					memDc.MoveTo(*(cp));
					memDc.LineTo(*(cp + 1));
				}
			}
		}

		else if ((*it)->type == 3) {
			MyStar* tmp = (MyStar*)(*it);

			//		polygon을 이용해야 외곽선을 칠할 수 있음
			memDc.Polygon(tmp->starList, 10);
		}

		//		외곽선
		memDc.SelectStockObject(NULL_BRUSH);
		CPen selPen(PS_DOT, 1, RGB(255, 0, 0));
		memDc.SelectObject(selPen);
```

```
                if ((*it)->isSel) {
                        memDc.Rectangle((*it)->selBox);
                }
        }
```

물론 buff에 저장한 도형도 출력해야 합니다. (위와 같은 알고리즘으로 순서를 지키기 위해 후순으로 출력)

이제 선택하는 알고리즘을 구현해보겠습니다. 고려해야 하는 것은 3가지입니다. 첫째, 빈 공간을 기준으로 드래그되었는지 여부, 둘째, Shift가 눌렸는지 여부, 셋째, 도형의 위치를 제대로 선택했는지 여부입니다.

먼저 도형의 위치를 선택했을 때입니다. 원과 사각형, 곡선은 확인하는 방법이 어렵지 않았지만 별의 내부를 확인하는 것이 가장 어려웠습니다. 이때 사용한 알고리즘은 CCW 알고리즘입니다. (from 멀티미디어 프로그래밍) 각 점과 한 점의 외적의 값들에 의해서 내부의 점인지 판별되는 알고리즘입니다. 하지만 이 알고리즘은 볼록 다각형의 기준으로만 작동한다는 것을 알게 되었고, 별을 총 6조각으로 나누었습니다. 가운데 오각형 1개, 주변 삼각형 5개, 총 6개의 도형 중 클릭한 지점이 하나라도 들어온다면 그 점은 그 별을 선택했다고 할 수 있는 것입니다. 아래는 그 알고리즘의 구현을 보여줍니다.

```
bool CChildView::isLeft(CPoint p, CPoint q, CPoint r) {
        float ax = q.x - p.x;
        float ay = q.y - p.y;
        float bx = r.x - p.x;
        float by = r.y - p.y;

        float cross = ax * by - ay * bx;

        if (cross >= 0) return false;
        return true;
}
bool CChildView::isStar(CPoint in, CPoint pt[]) {
        bool rst = false;
        for (int i = 1; i < 11; i += 2) {    //          겉에 있는 삼각형 5개
                if (!isLeft(pt[(i + 1) % 10], pt[(i + 2) % 10], in)) {
                        if (!isLeft(pt[(i + 2) % 10], pt[(i + 3) % 10], in)) {
                                if (!isLeft(pt[(i + 3) % 10], pt[(i + 1) % 10], in)) {
                                        rst = true;
                                }
                        }
                }
        }

        if (!rst) {    //          내부 확인
                rst = true;
```

```
                for (int i = 1; i < 11; i += 2) {
                        if (isLeft(pt[(i + 1) % 10], pt[(i + 2) % 10], in)) {
                                rst = false;
                        }
                }
        }
        return rst;
}
```

두 번째는 shift와 동시에 클릭을 했을 때입니다. 이때는 shift를 누르지 않았다면, 배열 내의 모든 도형을 선택하지 않게끔, 즉 초기화를 시킵니다.

```
if (!(nFlags & MK_SHIFT)||!isSel) {
                //      select shapeList 초기화
                isSel = false;
                list<MyShape*>::iterator it;
                for (it = shapeList.begin(); it != shapeList.end(); it++) {
                        (*it)->isSel = false;
                }
        }
```

Shift키가 눌리지 않았는지 확인하는 동시에 선택된 도형이 있는지 확인하는 isSel을 함께 사용해 판별합니다.

마지막으로 빈 곳을 클릭 후 드래그 하는 형태입니다. 이때 주의해야 할 점은 Shift키를 누른채로 드래그를 할 수 있다는 것입니다. 빈 곳을 선택했는지 isIn()함수를 이용해서 판별하고, 맞다면 클릭한 지점은 selPoint에 저장한 뒤에 드래그 한 영역에 도형들이 들어오는지 크기 비교를 통해 알아냅니다. (곡선과 별은 최대 x, y, 최소 x, y를 이용해서 Border가 갖춰져 있습니다.) 아래는 판별이 완료된 후에 선택하는 알고리즘까지 입니다.

```
//      Select Region 확정
selPoint2 = point;
SetCapture();
Invalidate();

// 중략

if (!isIn(point)) {
        if (selPoint != CPoint(0, 0) && selPoint2 != CPoint(0, 0)) {
                //      swap (어느 방향으로든 선택 가능)
                if (selPoint.x > selPoint2.x) swap(selPoint.x, selPoint2.x);
                if (selPoint.y > selPoint2.y) swap(selPoint.y, selPoint2.y);
                list<MyShape*>::iterator it;
                for (it = shapeList.begin(); it != shapeList.end(); it++) {
                        CPoint i = (*it)->invb;
                        CPoint b = (*it)->bor;
                                //      Select Box 안에 들어오는 객체들 선택
                        if (selPoint.x <= i.x && b.x <= selPoint2.x) {
```

```
                                        if (selPoint.y <= i.y && b.y <= selPoint2.y) {
                                                isSel = true;
                                                (*it)->isSel = true;
                                        }
                                }
                                Invalidate();
                        }
                }
}
```

도형의 둘레는 도형의 Border로 만들어 놓은 CRect를 통해 구현합니다. Margin값으로 5를 부여해서 만들고 붉은 점선으로 그려지게 끔 했습니다.

삭제는 list의 erase()를 사용하여 구현하였고, bring Front와 bring Back은 선택한 도형들을 메인 리스트에서 뽑아낸 후에 push_front() 혹은 push_back()을 하여 구현하였습니다. 따로 코드는 기술하지 않겠습니다.

우클릭하면 나오는 컨텍스트 메뉴와 메뉴와 툴바 모두 수업 내용을 바탕으로 제작하였기에 따로 기술하지 않겠습니다.

마지막으로 그룹핑입니다. 사실 가장 까다롭고 구현하기 어려웠던 알고리즘이었습니다. 그룹핑을 하는 것도 중요하지만, 그룹 내의 모든 원소를 출력, 어떠한 도형을 터치했을 때 그룹에 속했는지 혹은 어떤 그룹에 속했는지를 판별하는 것을 구현하기가 까다로웠습니다. 위에서 설명했듯이 그룹은 MyShape*를 만들어서 메인 리스트에 저장하고, 그룹으로 들어간 도형들은 메인 리스트에서 erase됩니다. 먼저 선택한 도형들 혹은 그룹들을 메인 리스트에서 뽑아내어 배열에 저장하고 저장한 도형들 혹은 그룹들을 메인 리스트에서 지워줍니다. 그 후에 배열에 저장한 도형들 혹은 그룹들을 MyShape*로 묶어 메인 리스트에 다시 저장합니다. 아래는 구현 코드입니다.

```
void CChildView::OnActionGroup()
{
        list<MyShape*>::iterator it;
        vector<MyShape*> group;

        for (it = shapeList.begin(); it != shapeList.end();) {
                if ((*it)->isSel) {
                        group.push_back(*it);
                        shapeList.erase(it);
                        it = shapeList.begin();
                }
                else {
                        it++;
                }
        }

        shapeList.push_back(new MyShape(true, group));
```

```
		isMore = false;

		group.clear();
		Invalidate();
}
```

그렇다면 언그룹핑은 그룹의 객체들을 메인 리스트에 넣고, 그룹 자체를 지워버리면 간단하게 구현 가능합니다. 이제 재귀함수들을 구현해야 합니다.

먼저 그룹 내의 모든 객체들을 그리는 Method입니다. 재귀함수를 통해 남기지 않고 모든 객체를 그려냅니다.

```
void CChildView::printAll(MyShape* G, CDC* memDc) {
		if (G->type == 4 && G->group.size() == 0) return;
		vector<MyShape*>::iterator it;

		for (it = G->group.begin(); it != G->group.end(); it++) {
			//		재귀함수
			if ((*it)->isG) {
				printAll((*it), memDc);
			}
		// 후략 -> 앞의 도형 그리는 방법과 사실상 유사
		}
}
```

만일 그 객체가 그룹이라면 그룹이 아닐 때까지 함수를 반복해서 실행하게 됩니다.

다음은 그룹 내의 객체가 선택되었는지 확인하는 알고리즘입니다. 가장 최상단의 root 객체가 선택되게끔 짜기 위해서 void가 아닌 bool형을 함수 값으로 반환했습니다.

```
bool CChildView::selParent(MyShape* tmp, CPoint point) {
		if (tmp->type == 4 && tmp->group.size() == 0) return false;

		vector<MyShape*>::reverse_iterator it;

		for (it = tmp->group.rbegin(); it != tmp->group.rend(); it++) {

			// 중략 -> 앞의 선택하는 사실상 유사

			else if ((*it)->type == 4) {		//Group
				(*it)->parent->isSel = selParent(*it, point);
			}
		}

		return isSel;
}
```

코드를 보면 알 수 있듯이 도형이 선택되는지 모든 경우에 접근을 하고 그룹 내의 도형 중 하나

라도 선택되었을 때, true를 아니라면 false를 반환합니다. 이 때 맨 위에 있는 도형부터 선택되어야 하기에 reverse_iterator를 사용해주었습니다.

지금까지 클래스에 대한 세부적인 설정과 중요한 알고리즘 몇 가지를 살펴보았습니다.

c) 시행착오

시행착오가 아주 많았기 때문에 대표적인 것들만 간추리겠습니다.

1) 원래는 참조를 받으려 했었다.

메인 리스트에 포인터가 아닌 참조를 받아서 값의 접근에 용이성을 가져오려 했습니다만, 참조 변수 저장의 구현에 난항을 겪어 실패했습니다. 하지만 업캐스팅과 다운캐스팅을 사용하기 위해서는 포인터가 조금 더 수월했습니다. 더하여 다운캐스팅은 수업시간에도 제대로 사용할 수 없다면 사용하지 말라고 하셨지만, 데이터가 날아가지 않게 끔 잘 사용하면 유용할 수도 있겠다는 생각을 했습니다.

2) 별을 선택하기

원래는 CCW 알고리즘에 제한사항이 있다는 것을 인지하지 못했었습니다. 그러나 10개의 점에 대해서 CCW를 진행하면 가운데에 있는 오각형 영역만 선택되는 것을 보고 볼록 다각형에서는 작동하지 않는다는 것을 알게 되었습니다. 따라서 위의 방식대로 6개의 영역으로 나누어 구현하는 방식으로 진행했습니다.

3) 재귀 함수, 어디서 탈출해야 하는가

재귀함수를 사용할 때 가장 중요한 점은 함수의 종료 지점입니다. 이제껏 만들어봤던 트리 형태와는 확연히 다른 형태였고, 코드도 훨씬 복잡했기에 다양한 접근이 들어갔었습니다. 인자로 넘어온 포인터가 NULL일 때라던 지, 인자로 넘어온 포인터의 부모 포인터가 NULL이라던지 등으로 했지만, 가지고 있는 group이 없을 때 탈출하는 것이 가장 직관적이고 깔끔하게 탈출할 수 있는 아이디어였습니다.

4) 그룹화

원래 그룹화를 할 때 진짜 트리구조를 만들려 하니, 감이 잘 잡히지 않았습니다. 또한 새로운 클래스를 만드는 것도 위험도가 높다고 판단했습니다. 그리하여 기본 클래스 자체를 그룹으로 사용하는 것이 가장 직관적이라고 생각했고, 구현을 시작했습니다. 하지만 이 방법은 기본 클래스 자체가 너무 복잡해지고 원래 알고 있던 트리 구조와 그에 관련된 알고리즘을 구현하기가 불가능에 가까웠습니다. 절반의 시간을 투자한 결과 구현을 완료할 수 있었습니다.

5) 클릭

가장 간단하다고 생각했지만 가장 구현하기 어려웠던 Method입니다. 드래그, 클릭, Shift 이 세가

지의 조건을 생각하며 알고리즘을 짜기는 정말 쉽지 않았습니다. 그래서 다양한 boolean형 변수를 도입해서 사용했고, 위에는 나오지 않았지만 isIn(), isInAll()이라는 함수를 따로 만들어 판별하기도 하였습니다. 하지만 너무 무턱대고 짰던 탓에 코드가 굉장히 복잡하고 오류나 예외가 생기면 고치기 어렵게 되었습니다. 코드를 정리하는 데에도 시간이 많이 소요되었고, 앞으로는 미리 구상한 후에 코드를 짜는 것이 훨씬 좋은 알고리즘을 짤 수 있을 것이라고 생각했습니다.

마지막 C++과제를 하면서 객체 지향을 사용해 보고, 이에 대한 이해도가 아주 많이 올라 갈 수 있었던 과제였던 것 같습니다. 특히 업캐스팅과 다운캐스팅을 이용해서 트리 형태의 그룹을 만들고, 또한 그룹이나 각 도형들을 하나의 배열에 담을 수 있다는 것을 알고서 구현 시켜, 객체 지향에서의 상속과 다형성이 매우 중요하다는 것을 알게 되었습니다. 마지막까지 좋은 수업과 과제를 주셔서 감사합니다.

이상입니다.