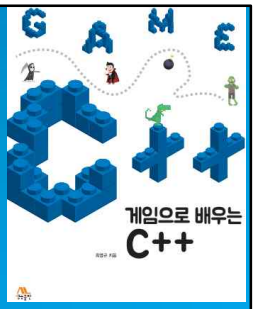
게임으로 배우는 C++

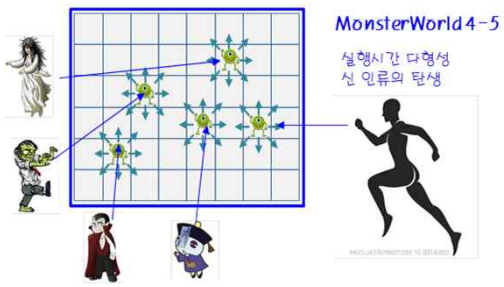


CHAPTER 10

다형성

몬스터월드4-5: 실행시간 다형성과 신일류의 탄생

- Monster World 4
  - 실행시간 다형성 적용



- Monster World 5
  - 신 인류의 탄생
  - 게임 기능
    - 키보드를 이용한 조작
    - 아이템을 먹기 위한 노력

2

10장 학습 목표

- 다형성의 종류와 실행시간 다형성의 의미를 이해한다.
- 상속에서의 상향 형 변환에 대해 이해한다.
- 가상 함수와 동적 바인딩을 이해한다.
- 상속에서 클래스의 크기가 어떻게 되는지를 이해한다.
- 가상 소멸자의 필요성을 이해한다.
- 순수 가상 함수와 추상 클래스를 이해한다.

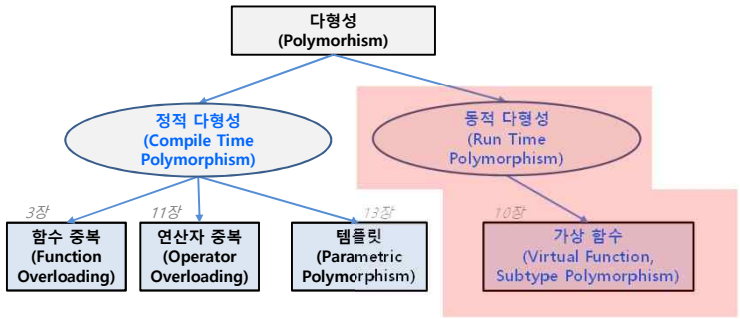
3

10.1 다형성이란?

- 다형성의 종류
- 동적 다형성이 필요한 이유

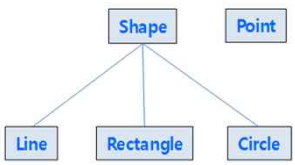
4

다형성의 종류



동적 다형성이 필요한 이유

- 그래픽 에디터 프로그램
  - 사용자가 그리는 객체의 순서를 알 수 없을 때



- 온갖 몬스터의 세상
  - 비슷한 코드가 계속 반복

10.2 응용: 상호작용이 가능한 그래픽 에디터

- 상호작용이 가능한 그래픽 에디터
- 객체의 저장과 관리 방법
- 실행 결과와 문제점

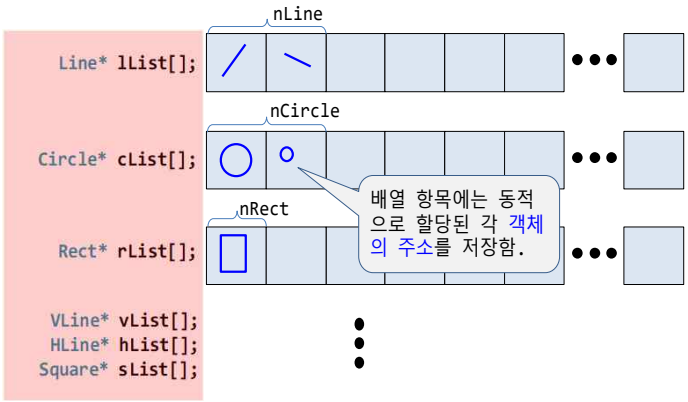
상호작용이 가능한 그래픽 에디터

- 실행시간에 다양한 객체를 선택해 입력 가능한 에디터
- 키보드를 이용한 객체 입력 방법

객체	의미	입력 방법
선분	(x1,y1)과 (x2,y2)를 양 끝점으로 하는 선분	**l** x1 y1 x2 y2 <엔터> *예) l 10 10 20 30 <엔터>*
사각형	시작 좌표가 (x,y)이고 가로가 width, 세로가 height인 사각형	**r** x y width height <엔터> *예) r 10 20 40 20 <엔터>*
원	중심이 (x,y)이고 반경이 radius인 원	**c** x y radius <엔터> *예) c 10 20 5 <엔터>*
수평선분	점 (x,y)을 시작으로 길이가 len인 수평 선분	**h** x y len <엔터> *예) h 10 20 5 <엔터>*
수직선분	점 (x,y)을 시작으로 길이가 len인 수직 선분	**v** x y len <엔터> *예) v 10 20 5 <엔터>*
정사각형	시작 좌표가 (x,y)이고 변의 길이가 len인 정사각형	**s** x y len <엔터> *예) s 10 20 5 <엔터>*
종료	프로그램 종료	**q** <엔터>

객체의 저장과 관리 방법

- 각 클래스마다 배열을 만들어 저장하는 방법

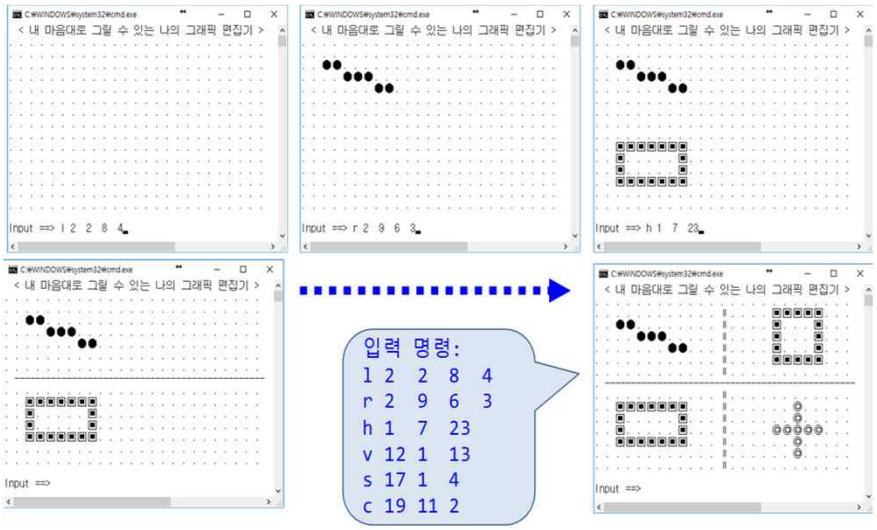


9

```
void main()
{
    Canvas myCanvas(25,15);
    Line* lList[100];
    VLine* vList[100];
    ...
    int  nLine = 0, nVLine = 0, nHLine = 0, ...;
    while (true) {
         ...
         printf("Input ==> ");
         gets(str);
         int ret = sscanf(str, "%c%d%d%d%d", &type, v,v+1,v+2,v+3);
         if (type == 'l' && ret == 5)
             lList[nLine++] = new Line(v[0], v[1], v[2], v[3]);
         else if (type == 'v' && ret == 4)
             vList[nVLine++] = new VLine(v[0], v[1], v[2]);
         ...
         else if (type == 'q') break;
         myCanvas.clear(". ");
         for (int i = 0; i < nLine; i++) lList[i]->draw(myCanvas,"●");
         for (int i = 0; i < nVLine; i++) vList[i]->draw(myCanvas,"‖");
         ...
    }
    for (int i = 0; i < nLine; i++) delete lList[i];
    for (int i = 0; i < nVLine; i++) delete vList[i];
    ...
}
```

10

실행 결과



11

문제점

- 사용자가 상호작용 할 수 있는 그래픽 편집기
  - 선, 사각형 및 원 객체를 저장할 배열을 각각 만듦.
  - 각 객체의 개수를 저장할 변수를 선언 및 0으로 초기화.
  - 사용자가 객체(선, 사각형, 원)를 추가할 때 마다 해당 클래스의 배열에 저장. 객체의 수 증가.
  - 각각의 배열을 모두 화면에 출력한다.

코드의 중복!
객체 입력 순서와 출력 순서가 다름!

- 해결 방안 ➔ 동적 다형성 사용
  - 먼저 형 변환을 이해해야 함.
  - 클래스의 자동 형 변환 이용

12

10.3 상속에서의 형변환?

- 기본 자료형의 형 변환
- 자동 상향 형 변환을 허용
- 명시적인 하향 형 변환
- Lab: 형 변환을 이용한 그래픽 에디터

13

기본 자료형의 형 변환

- 자동 형 변환

```
int x = 10;
double y = x;              // 자동 형 변환
int z = y;                 // 문제가 있는 형 변환
```

```
warning C4244: '초기화 중' : 'double'에서 'int'(으)로 변환하면서 데이터가 손실될 수 있습니다.
```

- 명시적 형 변환

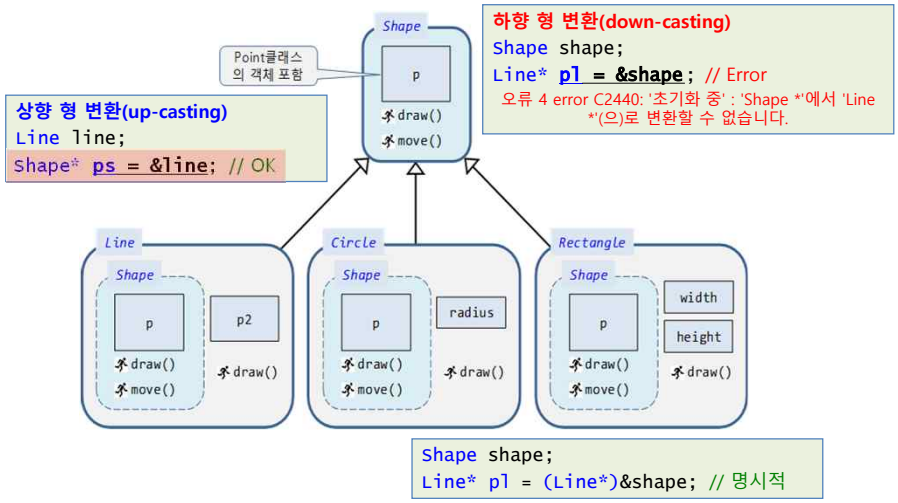
```
int z = (int)y;            // 명시적 형 변환
```

- 포인터에서의 형 변환

```
int x = 10;
int *pi = &x;
float *pf = (float*)pi;    // 포인터의 형 변환
*pf = 3.14f;               // 사용은 가능. 바람직하지 않음.
cout << *pf << endl;       // 3.14출력
cout << x << endl;         // 이상한 값이 출력됨
```

14

자동 상향 형 변환을 허용



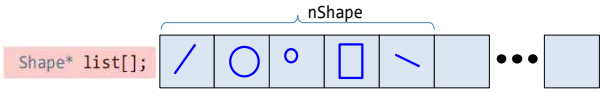
15

명시적인 하향 형 변환

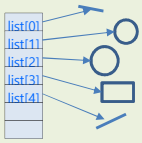
16

Lab: 형 변환을 이용한 그래픽 에디터

- 객체들의 관리 방법

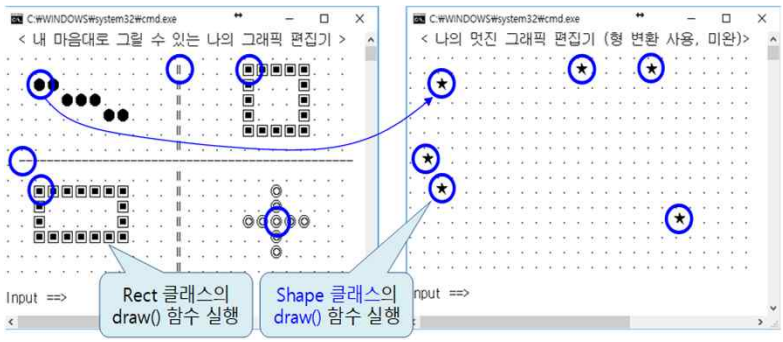



```
void main() {
    ...
    Shape*  list[100];
    int  nShape = 0;
    while (true) {
        ...
        if (type == 'l' && ret == 5)
            list[nShape++] = new Line(v[0], v[1], v[2], v[3]);
        ...
        else if (type == 'q') break;
        ...
        for (int i = 0; i < nShape; i++)
            list[i]->draw(myCanvas);
    }
    for (int i = 0; i < nShape; i++)
        delete list[i];
}
```



실행 결과???

- 출력 순서 ➔ 입력 순서와 동일
- 출력 내용 ➔ 모두 Shape의 draw()로 출력됨



- 원래 객체의 draw()함수 호출할 수는 없을까?
  - 동적 바인딩 사용. 실행시간 다형성.

10.4 가상 함수와 동적 바인딩

- 동적 바인딩
- 가상함수 선언 방법

19

동적 바인딩

- 동적 바인딩 (draw()함수)
  - Shape 클래스에서 draw()를 가상 함수로 선언.
  - 자식 클래스 Line, Circle 및 Rectangle에서 draw()를 재정의
    - 각 클래스의 특징에 따라 화면에 출력하는 함수를 구현.
  - 실행 시간에 list[i]->draw(); 문장을 만났을 때, draw()가
    - 가상 함수가 아니면: Shape 에서부터 draw() 를 찾음
    - 가상 함수이면: list[i]의 원래 클래스에서 draw() 를 찾음
- 가상함수 선언 방법

```
virtual void draw() { ... }
```

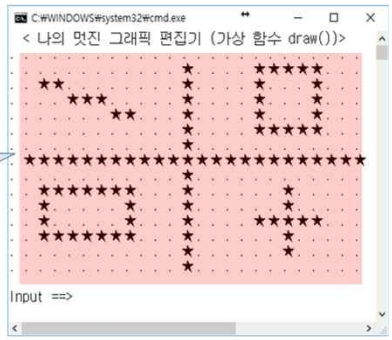
  - Shape의 멤버함수 draw() 앞에 virtual만 넣어주면 됨.

20

실행 결과 → OK

- 출력 순서 ➔ 입력 순서와 동일
- 출력 내용 ➔ 각 객체가 속한 클래스의 draw() 호출

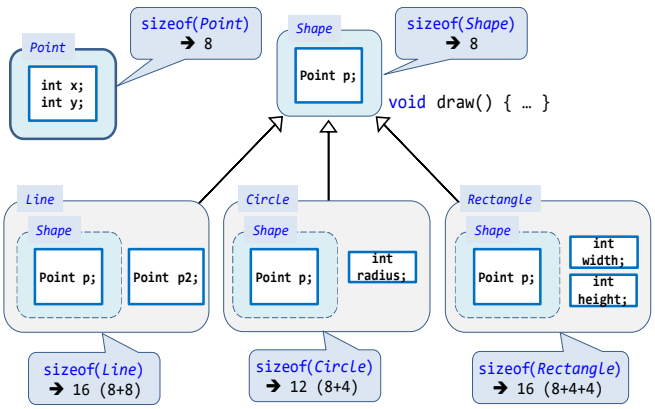
Shape의 draw()를 가상 함수로 선언함. 따라서 실행시간에 객체마다 원래 생성된 클래스의 draw() 함수가 실행됨.



10.5 상속에서의 객체 크기

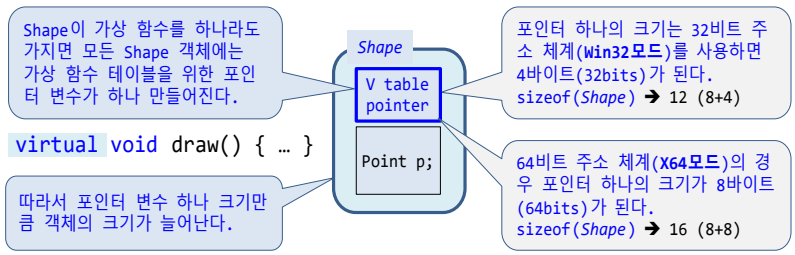
- 예: 그래픽 에디터의 클래스들
- 가상 함수와 객체의 크기
- 프로젝트 플랫폼별 차이
- 참조자와 다형성

예: 그래픽 에디터의 클래스들



가상 함수와 객체의 크기

- 가상 함수가 있는 클래스의 크기



프로젝트 플랫폼별 차이

- 포인터 크기의 차이: win32, x64

참조자와 다형성

- 참조자도 동일하게 동작됨

```
for (int i = 0; i < nShape; i++)
  list[i]->draw();
```

- 참조자 사용

```
for (int i = 0; i < nShape; i++) {
    Shape& s = *list[i];
    s.draw();
}
```

10.6 가상 소멸자

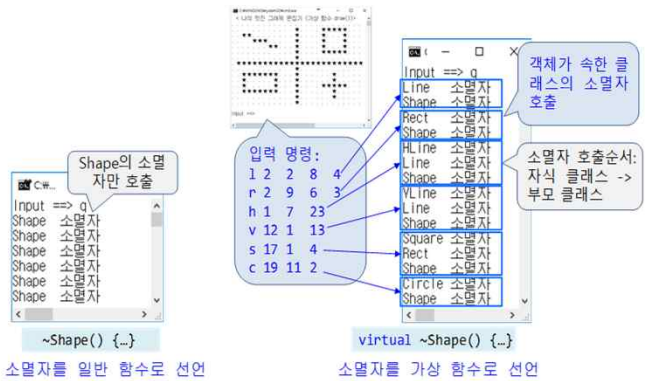
- 가상 소멸자란?

가상 소멸자란?

- 가상함수를 하나라도 사용한다면 소멸자도 가상 함수로.

```
virtual ~Shape() { ... }
```

- 가상 소멸자 사용에 따른 소멸자 호출 비교



10.7 응용: MonsterWorld 4

- Monster World 4: 실행시간 다형성

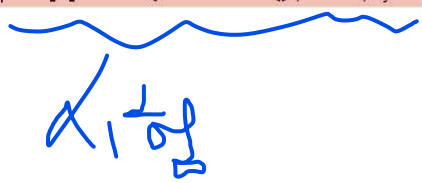
29

Monster World 4: 실행시간 다형성

- Monster의 소멸자와 move()함수를 가상함수로 선언함.
  - 코드의 중복을 없앰
  - 다시 반복문 사용 가능

```
((Zombie*)pMon[0])->move(world.Data(), xMax, yMax);
((Vampire*)pMon[1])->move(world.Data(),xMax, yMax);
...
```

```
for (int k = 0; k < nMon ; k++)
        pMon[k]->move(world.Data(), xMax, yMax);
```



30

10.8 순수 가상 함수와 추상 클래스

- 순수 가상 함수
- 추상 클래스
- 예: 동물 클래스들의 계층도

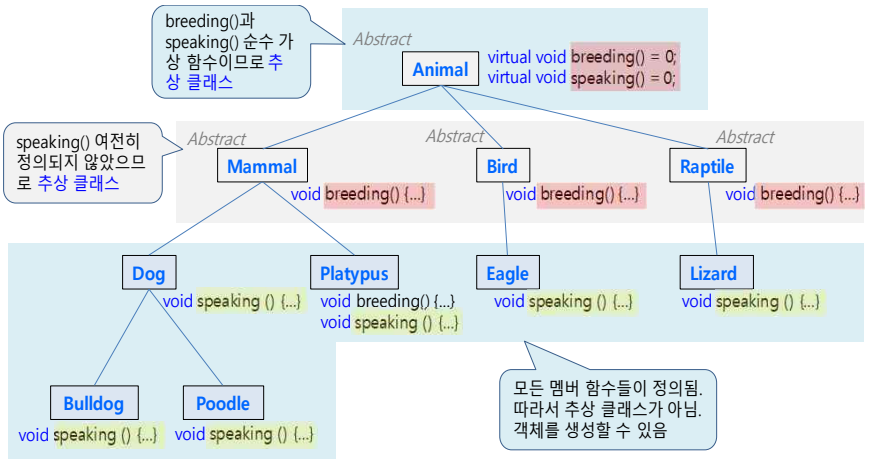
순수 가상 함수와 추상 클래스

- 순수 가상 함수의 필요성
  - 설계의 목적으로 함수 선언. 부모 클래스에서 구현하지는 않음
  - 헤더만 존재하고 몸체가 없는 함수

```
virtual 반환형 함수이름(매개변수리스트) = 0;
```

- 추상 클래스 (abstract class)
  - 멤버 함수들 중에 순수 가상 함수가 하나라도 있는 클래스
  - 객체를 생성할 수 없음.
    - `Shape s;  // Error`
  - 포인터 변수는 상관 없음
    - `Shape* ps;  // OK`

예: 동물 클래스들의 계층도



- 동물 클래스들의 계층도

```
Bulldog  mungchi;          // OK
Poodle   noeul;            // OK
dog      bato;             // OK
Mammal   poyou;            // Error
Mammal*  pm = &noeul;      // OK
Animal&  myPet = bato;     // OK
```

```
Animal*  aList[5];         // OK
aList[0] = new Eagle;      // OK
aList[1] = new Bird;       // Error
aList[2] = new Lizard;     // OK
aList[3] = new Poodle;     // OK
aList[4] = new Dog;        // OK
```

10.9 응용: MonsterWorld 5

- Monster World 5: 신인류의 탄생
- Human 클래스
- 고찰

Monster World 5: 신인류의 탄생

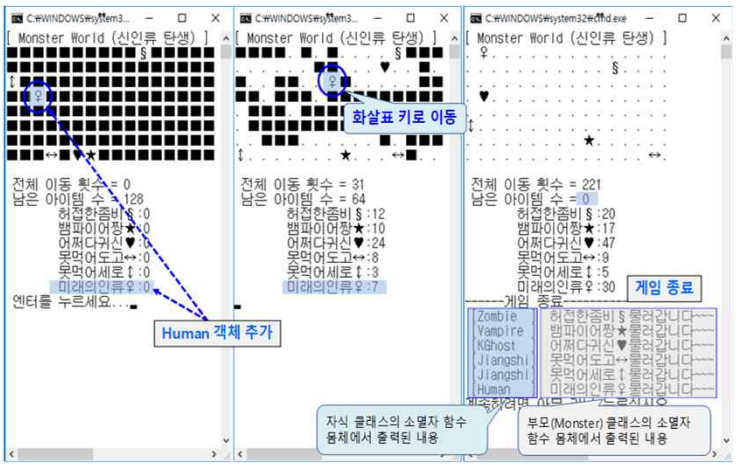
- 게이머가 참여하는 몬스터 월드
- Human 클래스 추가
  - "인류"는 게이머의 의지에 따라 키보드를 이용해 움직임
  - 키보드를 열심히 빨리 누를수록 더 많은 아이템을 먹게 됨
- 구현 방법
  - Monster 클래스를 상속해서 구현
  - move()함수를 재정의
  - 문제점: 게이머가 키를 입력하지 않더라도 다른 몬스터는 열심히 세상을 돌아다닐 수 있어야 한다.
  - 해결책: 키가 눌렸는지 검사하는 함수 사용
    - kbhit()

Human 클래스

```
class Human : public Monster{
public:
  Human(string n="미래인류", string i="♀", int px=0, int py=0)
        : Monster(n, i, px, py) {}
  ~Human() { cout << " [Human ]"; }
  int getDirKey() { return getche() == 224 ? getche() : 0; }
  void move(int** map, int maxx, int maxy) {
    if (kbhit()) {
        char ch = getDirKey();
        if (ch == Left) x--;
        else if (ch == Right) x++;
        else if (ch == Up) y--;
        else if (ch == Down) y++;
        else return;
        clip(maxx, maxy);
        eat(map);
    }
  }
};
```

37

실행 결과



38

고찰

- 게이머가 열심히 화살표를 움직이면 움직인 만큼의 아이템을 먹을 수 있음
- 게임이 종료되면 객체들이 소멸되는데, 자식의 소멸자가 먼저 호출된 다음 부모의 소멸자가 호출된 것을 확인
- 항상 처녀 귀신이 아이템 먹는데 유리함. 왜? 개선방안?
- 두 사람이 경쟁할 수 있도록 구현하는 방법은?

39

10장 요약문제, 연습문제, 실습문제

40

감사합니다!

41