**Homework 6**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.10.13.수 |

|  |
| --- |
| **6.1 가상함수와 지연 바인딩 (late-binding)을 사용하여 제공되는 다형성 (polymorphism)의 장점에 대하여 예를 들어 상세하게 설명하라.**   * 유사한 객체들의 관리 용이성 동일한 기반클래스로부터 상속받아 생성된 파생클래스의 객체들을 하나의 체계로 쉽게 관리할 수 있다.  기반클래스의 포인터 배열에 파생클래스의 포인터를 저장하고 하나의 체계 안에서 각 포인터에 대해 가상함수를 호출하고 late binding 시킴으로써 유사한 객체들을 쉽게 관리할 수 있다. * 시스템 확장성 기존의 상속관계에서 함께 개발된 파생클래스 이외에 나중에 추가되는 파생클래스들도 동일한 체계로 관리 가능하다.  기반 클래스를 상속한 파생클래스를 추가하고 내부에 가상함수를 구현한다. 기반클래스의 설계 및 구현 단계에서는 파생클래스의 존재를 모르며, 가상함수가 아직 구현되지 않은 상태다. 가상함수를 통해 아직 구현되지 않은 파생클래스의 멤버함수로 연결될 수 있도록 구성할 수 있다. 이를 통해 새로 추가한 class도 기존의 class와 동일한 체계로 관리하여 시스템을 확장할 수 있다. |
| **6.2 가상함수를 사용하여 다형성 (polymorphism)을 제공하기 위한 지연 바인딩 (late-binding)이 실행되는 방법에 대하여 예를 들어 구체적으로 설명하라.**   * 가상함수 내부동작 컴파일러가 가상함수 테이블을 만들어 가상함수들의 binding을 할 경우 어떤식으로 연결될 지에 대한 정보를 가지게 한다. 각 가상함수에는 포인터가 할당되며, 해당 가상함수가 실행될 때 실제 실행되는 함수를 가리키게 된다.  가상함수를 포함하는 객체는 가상함수 테이블을 가리키는 포인터를 가진다. 객체의 가상함수가 실행될 때 가상함수 테이블을 통해 실제 실행되는 함수를 쉽게 찾을 수 있도록 관리한다. * 다형성 구현을 위한 가상함수 설정 및 구현 절차 |
| **6.3 가상함수와 지연 바인딩 (late-binding)을 사용하여 구현되는 다형성 (polymorphism)의 단점에 대하여 예를 들어 상세하게 설명하라.**  가상함수는 다형성을 구현할 수 있게 해주는 중요한 장점이 있다.  하지만, 중요한 단점도 있다.   * 단점 1) 가상함수 테이블을 구성하고, 사용해야 하므로 메모리가 추가로 사용된다.  (컴파일 단계에서 가상함수 테이블이 생성된다.) 2) 가상함수가 실행될 때, 실제 실행하는 함수를 가상함수 테이블을 사용하여 찾고, 연결시켜야 하기 때문에 추가 지연이 발생된다.  아래의 동작에서 위의 단점이 확인된다.     가상함수를 사용하지 않고 구현할 때의 장점은 아래와 같다.  1) 가상함수 테이블을 구성하고 사용하는데 쓰이는 메모리가 절약된다.  2) 가상함수의 실행에서 실제 실행하는 함수를 가상함수 테이블을 통해 찾고 연결시키는 데 시간을 쓰지 않아도 된다.  따라서, 반드시 필요한 경우에만 가상 함수를 사용해야한다. |
| **6.4 상속, 다형성 및 가상함수를 사용하는 다음 C++ 클래스를 구현하라 : shape, Rectangle, Circle, Triangle, Hexahedron, Cylinder, and Prism.** |
| /\* main.cpp \*/  /\* Description  \* 도형에 대한 클래스 상속  \* 가상함수  \* Programmed by J. H. Kim  \* Last updated : 2021-10-13 \*/  #include <fstream>  #include "Position.h"  #include "Shape.h"  #include "Triangle.h"  #include "Prism.h"  #include "Rect.h"  #include "Hexahedron.h"  #include "Circle.h"  #include "Cylinder.h"  using namespace std;  #define NUM\_SHAPES 7  int main() {  ofstream fout;  // 객체 생성 및 초기화  Shape s1(Position(1, 1), 0, RGB\_BLUE, "Blue Shape");  Triangle t1(Position(2, 2), 3, 4, 0, RGB\_RED, "Red Triangle");  Prism p1(Position(3, 3), 6, 7, 8, 0, RGB\_GREEN, "Green Prism");  Rect r1(Position(4, 4), 1, 2, 0, RGB\_YELLOW, "Yellow Rectangle");  Hexahedron hx1(Position(5, 5), 5, 6, 7, 0, RGB\_ORANGE, "Orange Hexahedtron");  Circle cir(Position(6, 6), 6, 0, RGB\_MAGENTA, "Magenta Circle");  Cylinder cyl(Position(7, 7), 8, 6, 0, RGB\_WHITE, "White Cylinder");    // 체계에 사용할 기반클래스 포인터 배열 생성  Shape \* shapes[NUM\_SHAPES];    fout.open("Output.txt");  if (fout.fail())  {  cout << "Fail to open an output file (Output.txt)" << endl;  exit(1);  }  // 생성한 객체 포인터를 저장  shapes[0] = &s1;  shapes[1] = &t1;  shapes[2] = &p1;  shapes[3] = &r1;  shapes[4] = &hx1;  shapes[5] = &cir;  shapes[6] = &cyl;  // 출력을 위한 체계  for (int i = 0; i < NUM\_SHAPES; i++)  shapes[i]->draw(fout);  fout.close();  return 0;  } // end of main() |
| /\* Position.h \*/  #ifndef P\_H  #define P\_H  #include <iostream>  using namespace std;  typedef struct Position{  int x;  int y;  Position(int px, int py) {  x = px;  y = py;  };  } Position;  #endif // !P\_H |
| /\* Shape.h \*/  #ifndef S\_H  #define S\_H  #include <iostream>  #include <iomanip>  #include <Windows.h>  #include <conio.h>  #include <string>  #include "Position.h"  using namespace std;  #define PI 3.141592  const COLORREF RGB\_RED = 0x000000FF;  const COLORREF RGB\_GREEN = 0x0000FF00;  const COLORREF RGB\_BLUE = 0x00FF0000;  const COLORREF RGB\_BLACK = 0x00000000;  const COLORREF RGB\_ORANGE = 0x0000A5FF;  const COLORREF RGB\_YELLOW = 0x0000FFFF;  const COLORREF RGB\_MAGENTA = 0x00FF00FF;  const COLORREF RGB\_CYAN = 0x0000FFFF;  const COLORREF RGB\_WHITE = 0x00FFFFFF;  class Shape { // 기반 클래스  friend ostream& operator<<(ostream&, Shape&); // 출력  public:  // 생성자  Shape(); // default constructor  Shape(string name);  Shape(Position pos, double ang, COLORREF color, string name);  // 가상함수  virtual ~Shape();  virtual void draw(ostream&);  void fprint(ostream&); // 출력  int get\_pos\_x() const { return pos.x; } // x 좌표 반환  int get\_pos\_y() const { return pos.y; } // y 좌표 반환  void set\_pos\_x(int x) { pos.x = x; } // x 좌표 설정  void set\_pos\_y(int y) { pos.y = y; } // y 좌표 설정  string getName() { return name; } // 이름 반환  void setName(string n) { name = n; } // 이름 설정  Shape& operator=(const Shape& s); // 대입연산자  // ostream& fprintRGB(ostream&, COLORREF);  protected:  Position pos;  double angle;  string name;  COLORREF color;  };  #endif |
| /\* Shape.cpp \*/  #include "Shape.h"  ostream& operator<<(ostream&fout, Shape& s) {  s.fprint(fout);  return fout;  }  Shape::Shape()  : pos({ 0, 0 }), angle(0), name("noname"), color(RGB\_BLACK) {  }  Shape::Shape(string name)  : pos({ 0, 0 }), angle(0), name(name), color(RGB\_BLACK) {  }  Shape::Shape(Position pos, double ang, COLORREF color, string name)  : pos(pos), angle(ang), color(color), name(name) {  // pos.Position(pos.x, pos.y);  }  Shape::~Shape() {  }  void Shape::draw(ostream& fout) {  fout << "Shape::draw() => ";  fprint(fout);  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << uppercase << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << ")]" << dec;  fout << endl << " ";  fout << endl;  }  void Shape::fprint(ostream& fout) {  fout << "[name(" << setw(20) << name << "), ";  fout << "pos(" << pos.x << ", " << pos.y << "), ";  // fout << "color(" << color << ")]";  }  Shape& Shape::operator=(const Shape& s) {  pos = s.pos;  angle = s.angle;  name = s.name;  color = s.color;  return \*this;  }  /\*ostream& fprintRGB(ostream& ostr, COLORREF color) {  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  ostr << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << ")";  return ostr;  }\*/ |
| /\* Triangle.h \*/  #ifndef TRIANGLE\_H  #define TRIANGLE\_H  #include "Shape.h"  class Triangle : public Shape { // 파생 클래스  friend ostream& operator<<(ostream &, Triangle &);  public:  // 생성자  Triangle();  Triangle(string name);  Triangle(Position pos, int b, int h, double ang, COLORREF color, string name);  ~Triangle(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ostream& fout);  void fprint(ostream&); // 출력  int getBase() { return base; } // 밑변 반환  int getHeight() { return tri\_height; } // 높이 반환  Triangle& operator=(const Triangle& tri); // 대입연산자  protected:  int base;  int tri\_height;  };  #endif |
| /\* Triangle.cpp \*/  #include "Triangle.h"  ostream& operator<<(ostream& fout, Triangle& t) {  t.fprint(fout);    return fout;  }  Triangle::Triangle()  : Shape(), base(0), tri\_height(0) {  }  Triangle::Triangle(string name)  : Shape(name), base(0), tri\_height(0) {    }  Triangle::Triangle(Position pos, int b, int h, double ang, COLORREF color, string name)  : Shape(pos, ang, color, name), base(b), tri\_height(h) {    }  Triangle::~Triangle() {  //cout << "Triangle::Destructor (" << name << ").₩n";  }  double Triangle::getArea() {  return (base / 2.0) \* tri\_height;  }  void Triangle::draw(ostream& fout) {  fout << "Triangle::draw() => ";  fprint(fout);  }  void Triangle::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "base(" << base << "), tri\_height(" << tri\_height << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "triangle area(" << getArea() << ")]";  fout << endl << endl;  }  Triangle& Triangle::operator=(const Triangle& tri) {  Shape::operator=(tri);  base = tri.base;  tri\_height = tri.tri\_height;  return \*this;  } |
| /\* Prism.h \*/  #ifndef PSM\_H  #define PSM\_H  #include "Triangle.h"  class Prism : public Triangle { // 파생 클래스  friend ostream& operator<<(ostream&, const Prism&); // 출력  public:  // 생성자  Prism();  Prism(string name);  Prism(Position pos, int b, int th, int ht, double ang, COLORREF color, string name);  //Cylinder(Cylinder &cyl);  ~Prism(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ostream& fout);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getBase() const { return base; } // 밑변 반환  int getTriHeight() { return tri\_height; } // 밑면 높이 반환  Prism& operator=(const Prism& right); // 대입연산자  protected:  int height;  };  #endif // !PSM\_H |
| /\* Prism.cpp \*/  #include <math.h>  #include "Prism.h"  ostream& operator<<(ostream& fout, Prism& p) {  p.fprint(fout);  return fout;  }  Prism::Prism() // default constructor  : Triangle("no\_name"), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Prism::Prism(string name)  : Triangle(name), height(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Prism::Prism(Position pos, int b, int th, int ht, double ang, COLORREF color, string name)  : Triangle(pos, b, th, ang, color, name), height(ht) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Prism::~Prism() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Prism::getArea() {  return base \* tri\_height + base \* height +  2 \* sqrt(base \* base / 4.0 + tri\_height \* tri\_height) \* height;  }  void Prism::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Prism::draw(ostream& fout) {  fout << "Prism::draw() => ";  fprint(fout);  }  Prism& Prism::operator=(const Prism& right) {  Shape::operator=(right);  base = right.base;  tri\_height = right.tri\_height;  height = right.height;  return \*this;  }  void Prism::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "base(" << base << "), tri\_height(" << tri\_height << "), ";  fout << "height(" << height << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "prism surface area(" << getArea() << "), ";  fout << "prism volume(" << base \* tri\_height \* height / 2.0 << ")]";  fout << endl << endl;  } |
| /\* Rect.h \*/  #ifndef R\_H  #define R\_H  #include "Shape.h"  class Rect : public Shape { // 파생 클래스  friend ostream& operator<<(ostream &, Rect &);  public:  // 생성자  Rect();  Rect(string name);  Rect(Position pos, int w, int l, double ang, COLORREF color, string name);  ~Rect(); // 소멸자  double getArea(); // 표면적 반환  // 가상함수 구현  virtual void draw(ostream&);  void fprint(ostream&); // 출력  int getWidth() { return width; } // 가로 반환  int getLength() { return length; } // 세로 반환  Rect& operator=(Rect& rec); // 대입연산자  protected:  int width;  int length;  };  #endif // !R\_H |
| /\* Rect.cpp \*/  #include "Rect.h"  ostream& operator<<(ostream& fout, Rect& r) {  r.fprint(fout);  return fout;  }  Rect::Rect()  : Shape(), width(0), length(0) {  }  Rect::Rect(string name)  : Shape(name), width(0), length(0) {  }  Rect::Rect(Position pos, int w, int l, double ang, COLORREF color, string name)  : Shape(pos, ang, color, name), width(w), length(l) {  }  Rect::~Rect() {  }  double Rect::getArea() {  return width \* length;  }  // 가상함수 구현  void Rect::draw(ostream& fout) {  fout << "Rect::draw() => ";  fprint(fout);  }  void Rect::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "width(" << width << "), length(" << length << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "rectangle area(" << getArea() << ")]";  fout << endl << endl;  }  Rect& Rect::operator=(Rect& rec) {  Shape::operator=(rec);  width = rec.width;  length = rec.length;  return \*this;  } |
| /\* Hexahedron.h \*/  #ifndef HXHD\_H  #define HXHD\_H  #include "Rect.h"  class Hexahedron : public Rect { // 파생 클래스  friend ostream& operator<<(ostream&, const Hexahedron&); // 출력  public:  // 생성자  Hexahedron();  Hexahedron(string name);  Hexahedron(Position pos, int w, int l, int ht, double ang, COLORREF color, string name);  //Cylinder(Cylinder &cyl);  ~Hexahedron(); // 소멸자  double getArea(); // 표면적  // 가상함수 구현  virtual void draw(ostream& fout);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getWidth() const { return width; } // 가로 반환  int getLength() { return length; } // 세로 반환  Hexahedron& operator=(const Hexahedron& right); // 대입연산자  protected:  int height;  };  #endif // !HXHD\_H |
| /\* Hexahedron.cpp \*/  #include <math.h>  #include "Hexahedron.h"  ostream& operator<<(ostream& fout, const Hexahedron& hxhd) {  return fout;  }  Hexahedron::Hexahedron() // default constructor  : Rect("no\_name"), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Hexahedron::Hexahedron(string name)  : Rect(name), height(0) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Hexahedron::Hexahedron(Position pos, int w, int l, int ht, double ang, COLORREF color, string name)  : Rect(pos, w, l, ang, color, name), height(ht) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Hexahedron::~Hexahedron() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Hexahedron::getArea() {  return 2 \* (width \* length + width \* height + length \* height);  }  void Hexahedron::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Hexahedron::draw(ostream& fout) {  fout << "Hexahedron::draw() => ";  fprint(fout);  }  void Hexahedron::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "width(" << width << "), length(" << length << "), ";  fout << "height(" << height << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "hexahedron surface area(" << getArea() << "), ";  fout << dec << "hexahedron volume(" << width \* length \* height << ")]";  fout << endl << endl;  }  Hexahedron& Hexahedron::operator=(const Hexahedron& right) {  Shape::operator=(right);  width = right.width;  length = right.length;  height = right.height;  return \*this;  } |
| /\* Circle.h \*/  #ifndef Circle\_H  #define Circle\_H  #include "Shape.h"  #define PI 3.141592  class Circle : public Shape { // 파생 클래스  friend ostream& operator<<(ostream&, Circle&); // 출력  public:  // 생성자  Circle();  Circle(string name);  Circle(Position pos, int r, double ang, COLORREF color, string name);  //Circle(Circle &tr);  ~Circle(); // 소멸자  double getArea(); // 면적 반환  // 파생 클래스에 맞는 가상 함수 구현  virtual void draw(ostream&);  void fprint(ostream&); // 출력  int getRadius() const { return radius; } // Radius 반환  void setRadius(int r) { radius = r; } // Radius 설정  Circle& operator=(const Circle& right); // 대입연산자  protected:  int radius;  };  #endif |
| /\* Circle.cpp \*/  #include "Circle.h"  ostream& operator<<(ostream& fout, Circle& c) {  c.fprint(fout);  return fout;  }  Circle::Circle()  : Shape(), radius(0) {    }  Circle::Circle(string name)  : Shape(name), radius(0) {    }  Circle::Circle(Position pos, int r, double ang, COLORREF color, string name)  : Shape(pos, ang, color, name), radius(r) {    }  //Circle(Circle &tr);  Circle::~Circle() {  //cout << "Circle::Destructor (" << name << ").₩n";  }  double Circle::getArea() {  return PI \* radius \* radius;  }  void Circle::draw(ostream& fout) {  fout << "Circle::draw() => ";  fprint(fout);  }  void Circle::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "radius(" << radius << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "circle area(" << getArea() << ")]" << dec;  fout << endl << endl;  }  Circle& Circle::operator=(const Circle& right) {  Shape::operator=(right);  radius = right.radius;  return \*this;  } |
| /\* Cylinder.h \*/  #ifndef CYLINDER\_H  #define CYLINDER\_H  #include "Circle.h"  class Cylinder : public Circle { // 파생 클래스  friend ostream& operator<<(ostream&, const Cylinder&); // 출력  public:  // 생성자  Cylinder();  Cylinder(string name);  Cylinder(Position pos, int r, int height, int ang, COLORREF color, string name);  //Cylinder(Cylinder &cyl);  ~Cylinder(); // 소멸자  double getArea(); // 넓이  // 가상함수 구현  virtual void draw(ostream& fout);  virtual void draw(); // // used for testing of late binding  void fprint(ostream&); // 출력  int getRadius() const { return radius; } // 반지름 반환  void setRadius(int r) { radius = r; } // 반지름 설정  Cylinder& operator=(const Cylinder& right); // 대입연산자  protected:  int height;  };  #endif |
| /\* Cylinder.cpp \*/  #include <math.h>  #include "Cylinder.h"  using namespace std;  ostream& operator<<(ostream& fout, const Cylinder& cyl) {    return fout;  }  Cylinder::Cylinder() // default constructor  : Circle("no\_name"), height(0) {  //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";  }  Cylinder::Cylinder(string name)  : Circle(name), height(0) {  //cout << "AngleArc::Consb tructor (" << name << ").₩n";  }  Cylinder::Cylinder(Position pos, int r, int ht, int ang, COLORREF color, string name)  : Circle(pos, r, ang, color, name), height(ht) {  //cout << "AngleArc::Constructor (" << name << ").₩n";  }  Cylinder::~Cylinder() {  //cout << "Cylinder::Destructor (" << name << ").₩n";  }  double Cylinder::getArea() {  return 2 \* PI \* radius \* radius + 2 \* PI \* radius \* height;  }  void Cylinder::draw() {  cout << "draw() of Cylinder";  fprint(cout);  }  void Cylinder::draw(ostream& fout) {  fout << "Cylinder::draw() => ";  fprint(fout);  }  void Cylinder::fprint(ostream& fout) {  Shape::fprint(fout);  fout << "radius(" << radius << "), height(" << height << "), ";  int red, green, blue;  red = (color & 0x000000FF);  green = (color & 0x0000FF00) >> 8;  blue = (color & 0x00FF0000) >> 16;  fout << hex << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "  << setw(3) << blue << "), " << dec;  fout << endl << " ";  fout << "cylinder surface area(" << getArea() << "), ";  fout << "cylinder volume(" << PI \* radius \* radius \* height << ")]";  fout << endl << endl;  }  Cylinder& Cylinder::operator=(const Cylinder& right) {  Shape::operator=(right);  radius = right.radius;  height = right.height;  return \*this;  } |
|  |