객체지향프로그래밍과 자료구조 (실습)

Lab 6. (보충설명) Inheritance, Polymorphism, Virtual Function, ConsolePixelDrawing

정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- class Color
- class Shape
- class Console Pixel Drawing
- Windows Graphic Library
  - MoveToEx(hdc, p1.x, p1.y, NULL);
  - LineTo(hdc, p2.x, p2.y);
  - Ellipse(hdc, x1, y1, x2, y2);
  - Polygon(hdc, points, num_poly);
- class Circle
- class Triangle
- class Rectangle
- class Polygon

가상함수와 다형성 기반의 픽셀단위 도형 그리기

◆ Windows 환경에서의 픽셀단위 도형 출력을 위한 함수

구분	함수 및 인수	기능 설명
환경 설정	GetConsoleWindow()	get Console Window
	GetDC()	get device context
	SetBkColor(hdc, color)	배경 색깔 설정
	SetTextColor(hdc, color)	문자 색깔 설정
픽셀단위 위치 이동	MoveToEx(hdc, x, y, NULL)	지정된 좌표 (x, y)로 이동
펜 관리	CreatePen()	펜을 생성
브러시 관리	CreateSolidBrush()	도형 채우기에 사용되는 브러시 생성
픽셀단위 색깔 설정	SetPixel(hdc, x, y, color);	지정된 좌표 (x, y)의 픽셀을 지정된 색깔로 표시
도형 그리기	LineTo(hdc, x, y)	현재 픽셀 위치로부터 지정된 좌표까지 선 그리기
	Ellipse(hdc, x1, y1, x2, y2);	타원형 그리기
	ArcTo(hdc, int left, int top, int right, int bottom, int xr1, int yr1, int xr2, int yr2);	원호 (arc) 그리기
	Polygon(hdc, p, n)	POINT p[num_points]로 지정된 꼭지점을 연결하는 다각형 그리기
문자 출력	TextOut()	문자열 출력

Library Functions for Pixel-Drawing in Windows

Library Functions for Pixel-Drawing in MS-Windows

- MoveToEx(hdc, x, y, NULL);
- LineTo(hdc, x, y);
- **Ellipse(hdc, x1, y1, x2, y2);**
- ArcTo(hdc, int left, int top, int right, int bottom, int xr1, int yr1, int xr2, int yr2);
- **Polygon(hdc, points, num_poly);**
- Polyline(hdc, points, num_points);
- PolylineTo(HDC hdc, const POINT *lppt, DWORD cCount);

Library Functions for Pixel-Drawings on MS-Windows

Function	Description
AngleArc()	Draws a line segment and an arc defined by start angle and sweep angle
Arc()	Draws an elliptical arc.
ArcTo()	Draws an elliptical arc defined by bounding rectangle and start/end positions.
GetArcDirection()	Retrieves the current arc direction for the specified device context.
LineDDA()	Determines which pixels should be highlighted for a line defined by the specified starting and ending points.
LineDDAProc()	An application-defined callback function used with the LineDDA function.
LineTo()	Draws a line from the current position up to, but not including, the specified point.
MoveToEx()	Updates the current position to the specified point and optionally returns the previous position.

Library Functions for Pixel-Drawing in Windows

Function	Description
PolyBezier()	Draws one or more Bézier curves.
PolyBezierTo()	Draws one or more Bézier curves.
PolyDraw()	Draws a set of line segments and Bézier curves.
Polyline()	Draws a series of line segments by connecting the points in the specified array.
PolylineTo()	Draws one or more straight lines.
PolyPolyline()	Draws multiple series of connected line segments.
SetArcDirection()	Sets the drawing direction to be used for arc and rectangle functions

Data Structures for Pixel-based Drawing in Windows

- **#include <windows.h>**
- **HWND, HDC**
  - HWND console = GetConsoleWindow();
  - HDC hdc = GetDC(console); // dev_context
- **POINT**
  - POINT p; // p.x, p.y
  - POINT p[N];
- **HPEN, HBRUSH**
  - HPEN pen = CreatePen(PS_SOLID, pen_thickness, pen_color);
  - HBRUSH brush = CreateSolidBrush(brush_color);

ConsolePixelFrame.h

```
/* ConsolePixelDrawing.h (1) */
#ifndef PIXEL_DRAWING_H
#define PIXEL_DRAWING_H

#include <iostream>
#include <string>
#include <Windows.h>
#include <conio.h>
#include "Shape.h"
#include "Color.h"

using namespace std;

/* PEN_Stypes */
#define PS_SOLID         0
#define PS_DASH          1       // -------
#define PS_DOT           2       // .......
#define PS_DASHDOT       3       // _._._._
#define PS_DASHDOTDOT    4       // _.._.._
#define PS_NULL          5
#define PS_INSIDEFRAME   6
#define PS_USERSTYLE     7
#define PS_ALTERNATE     8

#define MAX_NUM_SHAPES 100
```

```
/* ConsolePixelDrawing.h (2) */

class Shape;

class ConsolePixelFrame
{
public:
    ConsolePixelFrame(int org_x, int org_y);
    ~ConsolePixelFrame();
    void addShape(Shape* new_shape);
    void drawShapes();
    int get_pos_org_x() { return pos_org_x; }
    int get_pos_org_y() { return pos_org_y; }
    HDC getConsole_DC() { return console_DC; }
private:
    HWND console;
    HDC console_DC; // device context
    Shape **pShapes; // Array of Shape Pointers
    int num_shapes;
    int capacity;
    int pos_org_x;
    int pos_org_y;
    bool isValidIndex(int sub);
};
#endif
```

ConsolePixelFrame.cpp

```
/* ConsolePixelDrawing.cpp (1) */
#include "ConsolePixelDrawing.h"

ConsolePixelFrame::ConsolePixelFrame(int px, int py)
{
    console = GetConsoleWindow();
    console_DC = GetDC(console);
    pShapes = new Shape*[MAX_NUM_SHAPES];
    num_shapes = 0;
    capacity = MAX_NUM_SHAPES;

    pos_org_x = px;
    pos_org_y = py;
}

ConsolePixelFrame::~ConsolePixelFrame()
{
     //delete[] shapes;

     //ReleaseDC(console, console_DC);
}
```

```
/* ConsolePixelDrawing.cpp (2) */

void ConsolePixelFrame::addShape(Shape *pNew_shape)
{
    if (num_shapes >= capacity)
    {
        cout << "ConsolePixelFrame::addShape ==> Expanding capacity to "
             << capacity * 2 << "shapes " << endl;
        Shape **old_shapes = pShapes;

        pShapes = new Shape*[capacity * 2];
        if (pShapes == NULL)
        {
            cout << "Error in expanding dynamic array for shapes capacity "
                 << capacity * 2 << "shapes " << endl;
            exit;
        }
        for (int i = 0; i < num_shapes; i++)
        {
            pShapes[i] = old_shapes[i];
        }
        capacity = capacity * 2;
        delete[] old_shapes;
    }
    pShapes[num_shapes] = pNew_shape;
    num_shapes++;
}
```

```
/* ConsolePixelDrawing.cpp (3) */

void ConsolePixelFrame::drawShapes()
{
    cout << "Drawing " << num_shapes << " shapes :" << endl;
    if (num_shapes > 0)
    {
        for (int i = 0; i < num_shapes; i++)
            pShapes[i]->draw(*this);
    }
}

bool ConsolePixelFrame::isValidIndex(int index)
{
     if ((index < 0) || (index >= num_shapes))
     {
        cout << "Error in ConsolePixelFrame::isValidIndex : current number of shapes ("
                << num_shapes << "), index : " << index << ") !!" << endl;
        return false;
     }
     else
        return true;
}
```

Standard Color Name and HexCode

Standard Color Name and RGB Code

1	AliceBlue	F0F8FF
2	AntiqueWhite	FAEBD7
3	Aqua	00FFFF
4	Aquamarine	7FFFD4
5	Azure	F0FFFF
6	Beige	F5F5DC
7	Bisque	FFE4C4
8	Black	000000
9	BlanchedAlmond	FFEBCD
10	Blue	0000FF
11	BlueViolet	8A2BE2
12	Brown	A52A2A
13	BurlyWood	DEB887
14	CadetBlue	5F9EA0
15	Chartreuse	7FFF00
16	Chocolate	D2691E
17	Coral	FF7F50
18	CornflowerBlue	6495ED
19	Cornsilk	FFF8DC
20	Crimson	DC143C
21	Cyan	00FFFF
22	DarkBlue	00008B
23	DarkCyan	008B8B
24	DarkGoldenRod	B8860B
25	DarkGray	A9A9A9
26	DarkGrey	A9A9A9
27	DarkGreen	006400
28	DarkKhaki	BDB76B
29	DarkMagenta	8B008B
30	DarkOliveGreen	556B2F
31	DarkOrange	FF8C00
32	DarkOrchid	9932CC
33	DarkRed	8B0000
34	DarkSalmon	E9967A
35	DarkSeaGreen	8FBC8F
36	DarkSlateBlue	483D8B
37	DarkSlateGray	2F4F4F
38	DarkSlateGrey	2F4F4F
39	DarkTurquoise	00CED1
40	DarkViolet	9400D3
41	DeepPink	FF1493
42	DeepSkyBlue	00BFFF
43	DimGray	696969
44	DimGrey	696969
45	DodgerBlue	1E90FF
46	FireBrick	B22222
47	FloralWhite	FFFAF0
48	ForestGreen	228B22
49	Fuchsia	FF00FF
50	Gainsboro	DCDCDC
51	GhostWhite	F8F8FF
52	Gold	FFD700
53	GoldenRod	DAA520
54	Gray	808080
55	Grey	808080
56	Green	008000
57	GreenYellow	ADFF2F
58	HoneyDew	F0FFF0
59	HotPink	FF69B4
60	IndianRed	CD5C5C
61	Indigo	4B0082
62	Ivory	FFFFF0
63	Khaki	F0E68C
64	Lavender	E6E6FA
65	LavenderBlush	FFF0F5
66	LawnGreen	7CFC00
67	LemonChiffon	FFFACD
68	LightBlue	ADD8E6
69	LightCoral	F08080
70	LightCyan	E0FFFF
71	LightGoldenRodYellow	FAFAD2
72	LightGray	D3D3D3
73	LightGrey	D3D3D3
74	LightGreen	90EE90
75	LightPink	FFB6C1
76	LightSalmon	FFA07A
77	LightSeaGreen	20B2AA
78	LightSkyBlue	87CEFA
79	LightSlateGray	778899
80	LightSlateGrey	778899
81	LightSteelBlue	B0C4DE
82	LightYellow	FFFFE0
83	Lime	00FF00
84	LimeGreen	32CD32
85	Linen	FAF0E6
86	Magenta	FF00FF
87	Maroon	800000
88	MediumAquaMarine	66CDAA
89	MediumBlue	0000CD
90	MediumOrchid	BA55D3
91	MediumPurple	9370DB
92	MediumSeaGreen	3CB371
93	MediumSlateBlue	7B68EE
94	MediumSpringGreen	00FA9A
95	MediumTurquoise	48D1CC
96	MediumVioletRed	C71585
97	MidnightBlue	191970
98	MintCream	F5FFFA
99	MistyRose	FFE4E1
100	Moccasin	FFE4B5
100	Moccasin	FFE4B5
101	NavajoWhite	FFDEAD
102	Navy	000080
103	OldLace	FDF5E6
104	Olive	808000
105	OliveDrab	6B8E23
106	Orange	FFA500
107	OrangeRed	FF4500
108	Orchid	DA70D6
109	PaleGoldenRod	EEE8AA
110	PaleGreen	98FB98
111	PaleTurquoise	AFEEEE
112	PaleVioletRed	DB7093
113	PapayaWhip	FFEFD5
114	PeachPuff	FFDAB9
115	Peru	CD853F
116	Pink	FFC0CB
117	Plum	DDA0DD
118	PowderBlue	B0E0E6
119	Purple	800080
120	RebeccaPurple	663399
121	Red	FF0000
122	RosyBrown	BC8F8F
123	RoyalBlue	4169E1
124	SaddleBrown	8B4513
125	Salmon	FA8072
126	SandyBrown	F4A460
127	SeaGreen	2E8B57
128	SeaShell	FFF5EE
129	Sienna	A0522D
130	Silver	C0C0C0
131	SkyBlue	87CEEB
132	SlateBlue	6A5ACD
133	SlateGray	708090
134	SlateGrey	708090
135	Snow	FFFAFA
136	SpringGreen	00FF7F
137	SteelBlue	4682B4
138	Tan	D2B48C
139	Teal	008080
140	Thistle	D8BFD8
141	Tomato	FF6347
142	Turquoise	40E0D0
143	Violet	EE82EE
144	Wheat	F5DEB3
145	White	FFFFFF
146	WhiteSmoke	F5F5F5
147	Yellow	FFFF00
148	YellowGreen	9ACD32

Color.h

```
/** Color.h */

#ifndef COLOR_H
#define COLOR_H
#include <Windows.h>
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

// COLORREF is defined in <Windows.h>
// The COLORREF value is used to specify an RGB color,
//   in hexadecimal form of 0x00bbggrr
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_WHITE = 0x00FFFFFF;
ostream& printRGB(ostream& ostr, COLORREF color);
// RGB color code chart: https://www.rapidtables.com/web/color/RGB_Color.html
/* Note: RGB(red, green, blue) macro also provides COLORREF data
    . RGB(FF, 00, 00) => 0x000000FF (RGB_RED)
    . RGB(00, FF, 00) => 0x0000FF00 (RGB_GREEN)
    . RGB(00, 00, FF) => 0x00FF0000 (RGB_BLUE)
*/
#endif
```

Color.cpp

```
/** Color.cpp */
#include <iostream>
#include <Windows.h>
#include "Color.h"

ostream& fprintRGB(ostream& fout, COLORREF color)
{
	int red, green, blue;

	red = (color & 0x000000FF);
	green = (color & 0x0000FF00) >> 8;
	blue = (color & 0x00FF0000) >> 16;
	fout << "RGB(" << setw(3) << red << ", " << setw(3) << green << ", "
		<< setw(3) << blue << ")";

	return fout;
}
```

class Shape

```
/* Shape.h (1) */

#ifndef SHAPE_H
#define SHAPE_H

#include <string>
#include <Windows.h>
#include <conio.h>
#include "ConsolePixelDrawing.h"
#include "Color.h"

using namespace std;
#define PI 3.14159

class ConsolePixelFrame;

class Shape
{
    friend ostream& operator<<(ostream &, Shape &);
public:
    Shape(); // default constructor
    Shape(string name);
    Shape(int px, int py, double angle, COLORREF ln_clr, COLORREF br_clr,
          int pen_thick, string name); // constructor
    virtual ~Shape();
```

```
/* Shape.h (2) */

    virtual void draw(ConsolePixelFrame cp_frame); // device context of console
    void fprint(ostream &);
    int get_pos_x() const { return pos_x; }
    int get_pos_y() const { return pos_y; }
    void set_pos_x(int x) { pos_x = x; }
    void set_pos_y(int y) { pos_y = y; }
    string getName() { return name; }
    void setName(string n) { name = n; }
    Shape& operator=(const Shape& s);
protected:
    int pos_x; // position x
    int pos_y; // position y
    double angle;
    string name;
    int pen_thickness;
    COLORREF line_color;
    COLORREF brush_color;
};

#endif
```

```
/** Shape.cpp (1) */
#include <iostream>
#include "Shape.h"
#include <iomanip>
using namespace std;

Shape::Shape()  // default constructor
{
    pos_x = pos_y = 0;
    angle = 0;
    line_color = brush_color = RGB_BLACK;
    name = "no_name";

    //cout << "Shape::Default constructor (" << name << ").₩n";
}

Shape::Shape(string n)
:name(n)
{
    pos_x = pos_y = 0;
    angle = 0;
    line_color = brush_color = RGB_BLACK;
    //cout << "Shape::Constructor (" << name << ").₩n";
}

Shape::~Shape()
{
    //cout << "Shape::Destructor (" << name << ").₩n";
}
```

```
/** Shape.cpp (2) */

Shape::Shape(int px, int py, double ang, COLORREF ln_clr, COLORREF br_clr,
    int pen_thick, string nm)
{
    pos_x = px;
    pos_y = py;
    angle = ang;
    line_color = ln_clr;
    brush_color = br_clr;
    pen_thickness = pen_thick;
    name = nm;
    //cout << "Shape::Constructor (" << name << ").₩n";
}

void Shape::fprint(ostream &fout)
{
    fout << name << ": pos(" << get_pos_x() << ", " << get_pos_y() << ")";
    fout << ", line_color(";
    fprintRGB(fout, line_color);
    fout << "), brush_color(";
    fprintRGB(fout, brush_color);
    fout << ")";
}

void Shape::draw(ConsolePixelFrame cp_frame)
{
    /* virtual function that will be late-binded to sub-class's draw() */
}
```

```
/** Shape.cpp (3) */
Shape& Shape::operator=(const Shape& s)
{
    pos_x = s.pos_x;
    pos_y = s.pos_y;
    angle = s.angle;
    name = s.name;
    line_color = s.line_color;
    brush_color = s.brush_color;

    return *this;
}

ostream& operator<<(ostream &ostr, Shape &shp)
{
    ostr << shp.name << ": pos(" << shp.get_pos_x() << ", " << shp.get_pos_y() << ")";
    ostr << ", line_color(";
    printRGB(ostr, shp.line_color);
    ostr << "), brush_color(";
    printRGB(ostr, shp.brush_color);
    ostr << ")";
    return ostr;
}
```

Pixel-based Drawing in Windows

◆ Move to Given Pixel Point and draw line

```
void drawLine(HDC hdc, POINT p1, POINT p2,
     int pen_thickness, COLORREF line_color)
{
     HPEN new_pen, old_pen;
     new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
     old_pen = (HPEN)SelectObject(hdc, new_pen);
     MoveToEx(hdc, p1.x, p1.y, NULL);
     LineTo(hdc, p2.x, p2.y);
     SelectObject(hdc, old_pen);
     DeleteObject(new_pen);
}
```

class Circle

```
/** Circle.h */

#include <string>
#include "Shape.h"
using namespace std;
#define PI 3.14159

class Circle : public Shape
{
    friend ostream& operator<<(ostream &, Circle &);
public:
    Circle();
    Circle(string name);
    Circle(Position pos, int r, int ang, SHAPE_COLOR clr, int pen_thick, string name);
    //Circle(Circle &cir);
    ~Circle();
    double getArea();
    void draw(ConsolePixelFrame cp_frame);
    void fprint(ostream &);
    int getradius() { return radius; }
    Circle& operator=(const Circle& cir);

protected:
    int radius;
};
```

```
void Circle::draw(ConsolePixelFrame cp_frame)
{
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    HDC hdc;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);
    Ellipse(hdc, pos_center_x - radius, pos_center_y - radius, pos_center_x + radius,
        pos_center_y + radius);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);

    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

class Triangle

```
/** Triangle.h */
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Triangle : public Shape
{
    //friend ostream& operator<<(ostream &, Triangle &);
public:
    Triangle();
    Triangle(string name);
    Triangle(Position pos, int b, int h, int ang, SHAPE_COLOR clr, int pen_thick, string name);
    //Triangle(Triangle &tr);
    ~Triangle();
    double getArea();
    virtual void draw(ConsolePixelFrame cp_frame);
    void fprint(ostream &);
    int getBase() { return base; }
    int getHeight() { return tri_height; }
    Triangle& operator=(const Triangle& tri);

protected:
    int base;
    int tri_height;
};
```

```
void Triangle::draw(ConsolePixelFrame cp_frame)
{
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    POINT p[3];
    p[0].x = pos_center_x - base / 2;
    p[0].y = pos_center_y + tri_height * 1.0 / 2.0;
    p[1].x = pos_center_x + base / 2;
    p[1].y = pos_center_y + tri_height * 1.0 / 2.0;
    p[2].x = pos_center_x;
    p[2].y = pos_center_y - tri_height * 1.0 / 2.0;
    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);

    Polygon(hdc, p, 3);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

class Rectangle

```
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Rectang : public Shape
{
   //friend ostream& operator<<(ostream &, Rectangle &);
public:
   Rectang();
   Rectang(string name);
   Rectang(Position pos, int w, int l, int ang, SHAPE_COLOR clr, int pen_thick, string name);
   //Rectangle(Rectangle &tr);
   ~Rectang();
   double getArea();
   virtual void draw(ConsolePixelFrame cp_frame);
   //void fprint(ostream &);
   int getWidth() { return width; }
   int getLength() { return length; }
   Rectang& operator=(Rectang& rec);

protected:
   int width;
   int length;
};
```

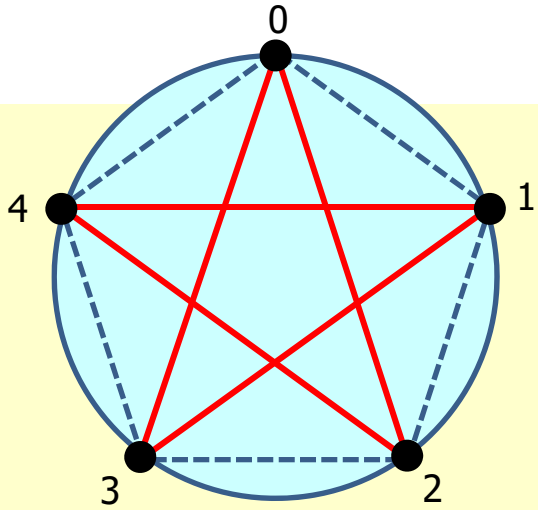
```
void Rectang::draw(ConsolePixelFrame cp_frame)
{
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    POINT p[4];
    p[0].x = pos_center_x - width / 2;
    p[0].y = pos_center_y - length / 2;
    p[1].x = pos_center_x + width / 2;
    p[1].y = pos_center_y - length / 2.0;
    p[2].x = pos_center_x + width / 2;
    p[2].y = pos_center_y + length / 2.0;
    p[3].x = pos_center_x - width / 2;
    p[3].y = pos_center_y + length / 2.0;

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(cp_frame.getConsole_DC(), new_brush);
    Polygon(hdc, p, 4);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

class PolyGon

```
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class PolyGon : public Shape
{
    //friend ostream& operator<<(ostream &, PolyGonle &);
public:
    PolyGon();
    PolyGon(string name);
    PolyGon(Position pos, int radius, int num_poly, int ang, SHAPE_COLOR clr, int pen_thick, string name);
    //PolyGonle(PolyGonle &pg);
    ~PolyGon();
    //double getArea();
    virtual void draw(ConsolePixelFrame cp_frame);
    //void fprint(ostream &);
    int getRadius() { return radius; }
    int getNumPoly() { return num_poly; }
    PolyGon& operator=(PolyGon& pg);

protected:
    int radius;
    int num_poly;
};
```

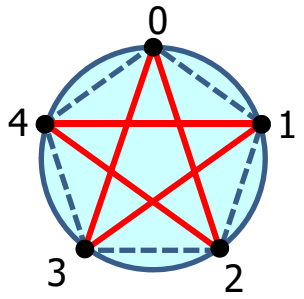
```
void PolyGon::draw(ConsolePixelFrame cp_frame)
{
    POINT *points = (POINT *)malloc(num_poly * sizeof(POINT));
    double rad_angle, delta_angle; // angle in radian
    int pos_center_x, pos_center_y;
    pos_center_x = cp_frame.get_pos_org_x() + get_pos_x();
    pos_center_y = cp_frame.get_pos_org_y() + get_pos_y();

    int x, y;
    HDC hdc;
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    hdc = cp_frame.getConsole_DC();

    delta_angle = 2.0*PI / num_poly;
    rad_angle = PI / 2.0;
    for (int i = 0; i < num_poly; i++, rad_angle += delta_angle)
    {
        x = pos_center_x + radius * cos(rad_angle);
        y = pos_center_y - radius * sin(rad_angle);
        points[i].x = x;  points[i].y = y;
    }

    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(cp_frame.getConsole_DC(), new_brush);
    Polygon(hdc, points, num_poly);
    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);
    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```

class Star




```
/* Star.h */
#ifndef Star_H
#define Star_H
#include <string>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
using namespace std;

class Star : public Shape
{
public:
        Star();
        Star(string name);
        Star(int px, int py, int radius, int num_vertices, double ang, COLORREF ln_clr,
            COLORREF br_clr, int pen_thick, string name);
        ~Star();
        virtual void draw(ConsolePixelFrame cp_frame);
        virtual void draw(); // // used for testing of late binding
        void fprint(ostream &);
        int getRadius() { return radius; }
        int getNumPoly() { return num_vertices; }
        Star& operator=(Star& pg);
protected:
        int radius;
        int num_vertices;
};

#endif
```

```
void Star::draw(ConsolePixelFrame cp_frame)
{
      POINT *points = (POINT *)malloc(5.0 *
         sizeof(POINT));
      double rad_angle, delta_angle;
         // angle in radian
      int pos_center_x, pos_center_y;
      pos_center_x = cp_frame.get_pos_org_x() +
         get_pos_x();
      pos_center_y = cp_frame.get_pos_org_y() +
         get_pos_y();

      int x, y;
      HDC hdc;
      HPEN new_pen, old_pen;
      HBRUSH new_brush, old_brush;
      hdc = cp_frame.getConsole_DC();

      delta_angle = 2.0*PI / 5.0;
      rad_angle = PI / 2.0;
      for (int i = 0; i < 5.0; i++, rad_angle +=
         delta_angle)
      {
            x = pos_center_x + radius * cos(rad_angle);
            y = pos_center_y - radius * sin(rad_angle);
            points[i].x = x;
            points[i].y = y;
      }
```



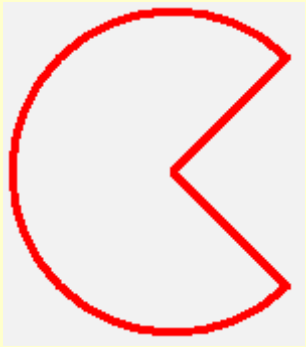

```
      new_pen = CreatePen(PS_SOLID,
          pen_thickness, line_color);
      old_pen = (HPEN)SelectObject(hdc,
          new_pen);
      new_brush = CreateSolidBrush(brush_color);
      old_brush = (HBRUSH)SelectObject(hdc,
          new_brush);

      //instead of Polygon(hdc, points, num_poly);
      MoveToEx(hdc, points[0].x, points[0].y,
          NULL);
      LineTo(hdc, points[2].x, points[2].y);
      LineTo(hdc, points[4].x, points[4].y);
      LineTo(hdc, points[1].x, points[1].y);
      LineTo(hdc, points[3].x, points[3].y);
      LineTo(hdc, points[0].x, points[0].y);

      SelectObject(hdc, old_pen);
      DeleteObject(new_pen);
      SelectObject(hdc, old_brush);
      DeleteObject(new_brush);
}
```

class AngledArc

```
/* AngleArc.h */
#ifndef ANGLE_ARC_H
#define ANGLE_ARC_H

#include <string>
#include "Shape.h"
using namespace std;

class AngledArc : public Shape
{
    friend ostream& operator<<(ostream&, const AngledArc&);
public:
    AngledArc();
    AngledArc(string name);
    AngledArc(int px, int py, int r, int ang, int start_ang, int sweep_ang, COLORREF ln_clr,
COLORREF br_clr, int pen_thick, string name);
    //AngledArc(AngledArc &angarc);
    ~AngledArc();
    virtual void draw(ConsolePixelFrame cp_frame);
    virtual void draw(); // // used for testing of late binding
    void fprint(ostream&);
    int getRadius() const { return radius; }
    void setRadius(int r) { radius = r; }
    AngledArc& operator=(const AngledArc& cir);

protected:
    int radius;
    int start_angle;
    int sweep_angle;
};

#endif
```

```
/** AngleArc.cpp (1) */
#include <iostream>
#include <math.h>
#include "AngledArc.h"
#include "ConsolePixelDrawing.h"
#include <iomanip>
using namespace std;

AngledArc::AngledArc() // default constructor
    :Shape("no_name"), radius(0), start_angle(0), sweep_angle(0)
{
    //cout << "AngleArc::Default AngleArc constructor (" << name << ").₩n";
}
AngledArc::AngledArc(string name)
    : Shape(name), radius(0), start_angle(0), sweep_angle(0)
{
    //cout << "AngleArc::Constructor (" << name << ").₩n";
}
AngledArc::AngledArc(int px, int py, int r, int ang, int start_ang, int sweep_ang,
     COLORREF ln_clr, COLORREF br_clr, int pen_thick, string name)
    : Shape(px, py, ang, ln_clr, br_clr, pen_thick, name)
{
    //cout << "AngleArc::Constructor (" << name << ").₩n";
    radius = r;
    start_angle = start_ang;
    sweep_angle = sweep_ang;
}
AngledArc::~AngledArc()
{
    //cout << "AngleArc::Destructor (" << name << ").₩n";
}
void AngledArc::draw()
{
    cout << "draw() of AngleArc";
    fprint(cout);
}
```

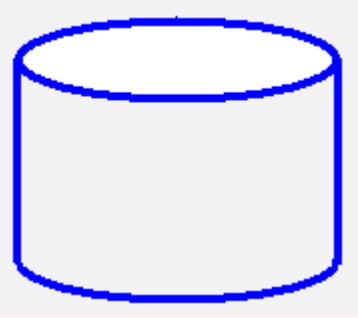
```
/** AngleArc.cpp (2) */
void AngledArc::draw(ConsolePixelFrame cp_frame)
{
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    HDC hdc;
    int fr_px, fr_py;
    int start_px, start_py;
    double start_ang_rad;

    fr_px = cp_frame.get_cpfr_px() + get_px();
    fr_py = cp_frame.get_cpfr_py() + get_py();

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);

    start_ang_rad = start_angle * 3.141592 / 180.0;
    start_px = fr_px + radius * cos(start_ang_rad);
    start_py = fr_py - radius * sin(start_ang_rad);
    MoveToEx(hdc, fr_px, fr_py, (LPPOINT)NULL);
    LineTo(hdc, start_px, start_py);
    AngleArc(hdc, fr_px, fr_py, radius, start_angle, sweep_angle);
    LineTo(hdc, fr_px, fr_py);

    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);

    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```



```
/** AngleArc.cpp (3) */


ostream& operator<<(ostream& fout, const AngledArc& ang_arc)
{
    fout << ang_arc.name << ": pos(" << ang_arc.get_px() << ", " << ang_arc.get_py() << ")";
    fout << ", line_color(";     fprintRGB(fout, ang_arc.line_color);
    fout << "), brush_color(";     fprintRGB(fout, ang_arc.brush_color);
    fout << ")";
    fout << ", radius(" << ang_arc.radius << ")";
    fout << ", start_ang (" << ang_arc.start_angle << "), sweep_ang(" << ang_arc.sweep_angle << ")";
    fout << endl;
    return fout;
}

AngledArc& AngledArc::operator=(const AngledArc& right)
{
    Shape::operator=(right);
    radius = right.radius;
    return *this;
}

void AngledArc::fprint(ostream& fout)
{
    Shape::fprint(fout);
    fout << ", radius (" << radius << "), start_angle (" << start_angle:
    fout << "), sweep_angle (" << sweep_angle << ")";
    fout << endl;
}
```

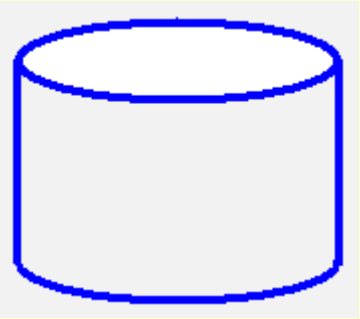
class Cylinder

```
/* Cylinder.h */
#ifndef CYLINDER_H
#define CYLINDER_H

#include <string>
#include "Shape.h"
using namespace std;

class Cylinder : public Shape
{
    friend ostream& operator<<(ostream&, const Cylinder&);
public:
    Cylinder();
    Cylinder(string name);
    Cylinder(int px, int py, int r, int ang, int height, COLORREF ln_clr, COLORREF br_clr,
        int pen_thick, string name);
    //Cylinder(Cylinder &cyl);
    ~Cylinder();
    double getArea();
    virtual void draw(ConsolePixelFrame cp_frame);
    virtual void draw(); // // used for testing of late binding
    void fprint(ostream&);
    int getRadius() const { return radius; }
    void setRadius(int r) { radius = r; }
    Cylinder& operator=(const Cylinder& cir);

protected:
    int radius;
    int height;
};

#endif
```



```
/* Cylinder.cpp (1) */

#include <iostream>
#include <math.h>
#include "Cylinder.h"
#include "ConsolePixelDrawing.h"
#include <iomanip>
using namespace std;

Cylinder::Cylinder() // default constructor
    : Shape("no_name"), radius(0), height(0)
{
    //cout << "Cylinder::Default AngleArc constructor (" << name << ").₩n";
}
Cylinder::Cylinder(string name)
    : Shape(name), radius(0), height(0)
{
    //cout << "AngleArc::Constructor (" << name << ").₩n";
}
Cylinder::Cylinder(int px, int py, int r, int ang, int ht, COLORREF ln_clr, COLORREF br_clr,
    int pen_thick, string name)
    : Shape(px, py, ang, ln_clr, br_clr, pen_thick, name)
{
    //cout << "AngleArc::Constructor (" << name << ").₩n";
    radius = r;
    height = ht;
}
Cylinder::~Cylinder()
{
    //cout << "Cylinder::Destructor (" << name << ").₩n";
}
void Cylinder::draw()
{
    cout << "draw() of Cylinder";
    fprint(cout);
}
```

```
/* Cylinder.cpp (2) */

void Cylinder::draw(ConsolePixelFrame cp_frame)
{
    HPEN new_pen, old_pen;
    HBRUSH new_brush, old_brush;
    HDC hdc;
    int center_px, center_py;
    int start_px, start_py;
    double start_ang_rad;

    center_px = cp_frame.get_cpfr_px() + get_px();
    center_py = cp_frame.get_cpfr_py() + get_py();

    hdc = cp_frame.getConsole_DC();
    new_pen = CreatePen(PS_SOLID, pen_thickness, line_color);
    old_pen = (HPEN)SelectObject(hdc, new_pen);
    new_brush = CreateSolidBrush(brush_color);
    old_brush = (HBRUSH)SelectObject(hdc, new_brush);

    MoveToEx(hdc, center_px - radius, center_py - height/2, (LPPOINT)NULL);
    LineTo(hdc, center_px - radius, center_py + height / 2);
    MoveToEx(hdc, center_px + radius, center_py - height / 2, (LPPOINT)NULL);
    LineTo(hdc, center_px + radius, center_py + height / 2);

    Ellipse(hdc, center_px - radius, center_py - height/2 - 20, center_px + radius,
       center_py - height/2 + 20);
    MoveToEx(hdc, center_px - radius, center_py + height / 2, (LPPOINT)NULL);
    ArcTo(hdc, center_px - radius, center_py + height/2 - 20, center_px + radius,
       center_py + height/2 + 20, center_px - radius, center_py + height/2, center_px + radius,
       center_py + height/2);

    SelectObject(hdc, old_pen);
    DeleteObject(new_pen);

    SelectObject(hdc, old_brush);
    DeleteObject(new_brush);
}
```



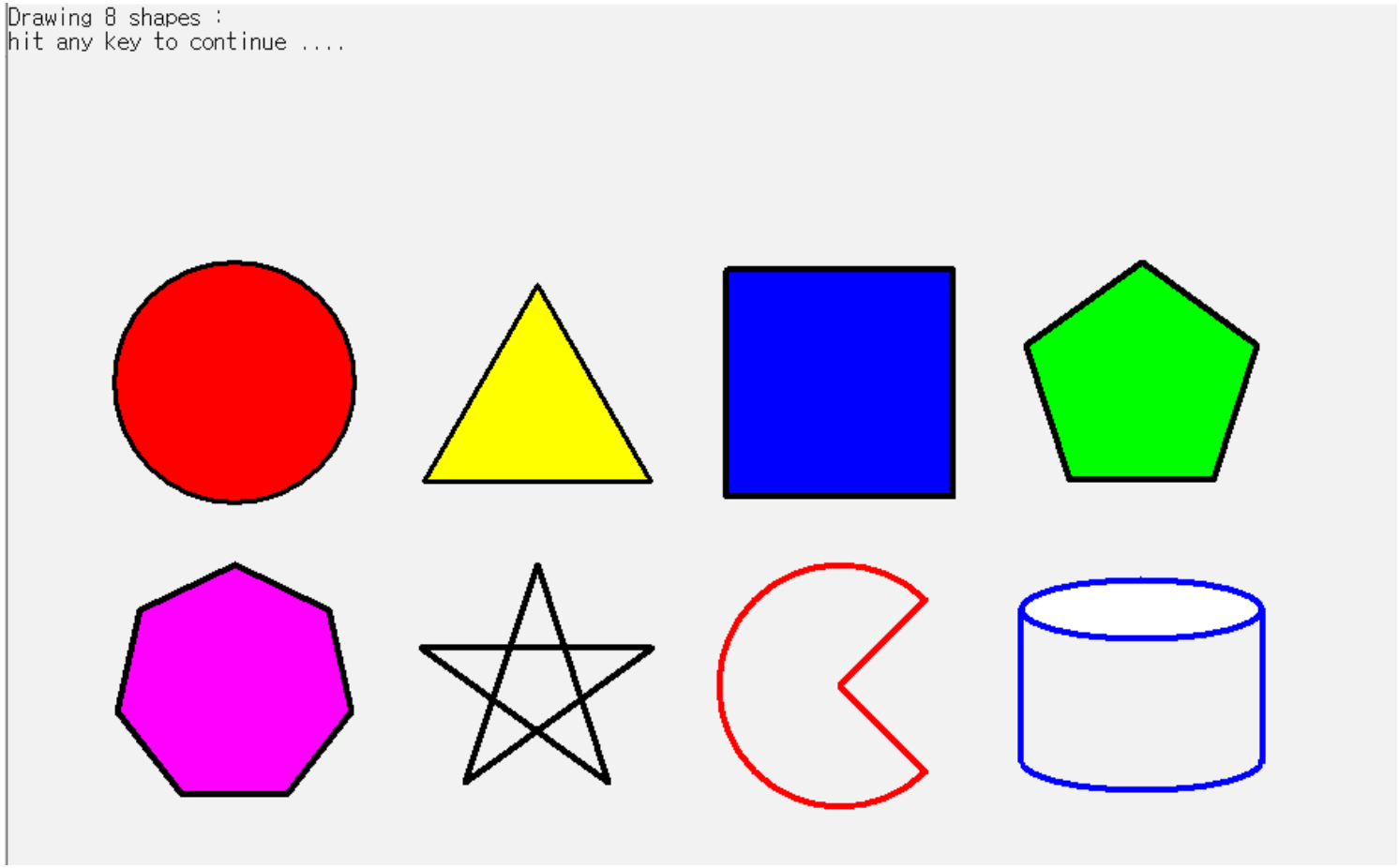
```
/* Cylinder.cpp (3) */

ostream& operator<<(ostream& fout, const Cylinder& cyl)
{
    fout << cyl.name << ": pos(" << cyl.get_px() << ", " << cyl.get_py() << ")";
    fout << ", line_color(";
    fprintRGB(fout, cyl.line_color);
    fout << "), brush_color(";
    fprintRGB(fout, cyl.brush_color);
    fout << ")";
    fout << ", radius(" << cyl.radius << ")";
    fout << ", height (" << cyl.height << ")";
    fout << endl;
    return fout;
}


Cylinder& Cylinder::operator=(const Cylinder& right)
{
    Shape::operator=(right);
    radius = right.radius;
    height = right.height;

    return *this;
}

void Cylinder::fprint(ostream& fout)
{
    Shape::fprint(fout);
    fout << ", radius (" << radius << "), height (" << height << ")";
    fout << endl;
}
```

main()

```
/** main.cpp  (1) */
#include <iostream>
#include <string>
#include <fstream>
#include "ConsolePixelDrawing.h"
#include "Shape.h"
#include "Triangle.h"
#include "Circle.h"
#include "Rectang.h"
#include "Polygon.h"
#include "Star.h"
#include "AngledArc.h"
#include "Cylinder.h"

using namespace std;
```

```
/** main.cpp  (2) */

int main()
{
    Circle  cir(100, 200, 80, 0, RGB_BLACK, RGB_RED, 3, "Circle");
    Triangle tri(300, 200, 150, 130, 0, RGB_BLACK, RGB_YELLOW, 3, "Triangle");
    Rectang rec(500, 200, 150, 150, 0, RGB_BLACK, RGB_BLUE, 4, "Rectangle");
    PolyGon poly_5(700, 200, 80, 5, 0, RGB_BLACK, RGB_GREEN, 4, "Polygon_5");
    PolyGon poly_7(100, 400, 80, 7, 0, RGB_BLACK, RGB_MAGENTA, 4, "Polygon_7");
    Star star_5(300, 400, 80, 5, 0, RGB_BLACK, RGB_GREEN, 4, "Star_5");
    AngledArc angle_arc(500, 400, 80, 0, 45, 270, RGB_RED, RGB_BLUE, 4, "Angle_Arc");
    Cylinder cyl(700, 400, 80, 0, 100, RGB_BLUE, RGB_WHITE, 4, "Cylinder");

    ConsolePixelFrame frame(50, 50); // fr_x, fr_y


    return 0;

}  // end of main()
```

Example Result



Lab 6. Oral Test

6.1 다형성 (polymorphism)이 무엇이며, 왜 필요한가에 대하여 예를 들어 설명하라.

6.2 다형성 (polymorphism)을 구현하기 위하여 사용되는 가상함수 (vritual function)와 지연 바인딩 (late binding)이 무엇이며, 어떻게 동작하는지에 대하여 예를 들어 설명하라.

6.3 가상함수와 late binding 기능을 사용하여 화면에 class Shape으로부터 상속받은 다수의 도형들을 class Shape의 포인터로 drawing하는 방법에 대하여 상세하게 설명하라.

6.4 Upcasting slicing이 어떤 문제이며, 왜 발생하는가에 대하여 예를 들어 설명하라.