

<5.6>

다음 프로그램의 실행 결과를 쓰시오.

(1)

```
using System;

//5.6 (1)

참조 2개
class BaseClass { public static int supNum = 100; }

참조 2개
class DerivedClass : BaseClass { public static int subNum = 200; }

참조 0개
public class Exercise5_6_1
{
    참조 0개
    public static void Main()
    {
        Console.Write(BaseClass.supNum + ", " + DerivedClass.supNum);

        Console.WriteLine(", " + DerivedClass.subNum);
    }
}
```

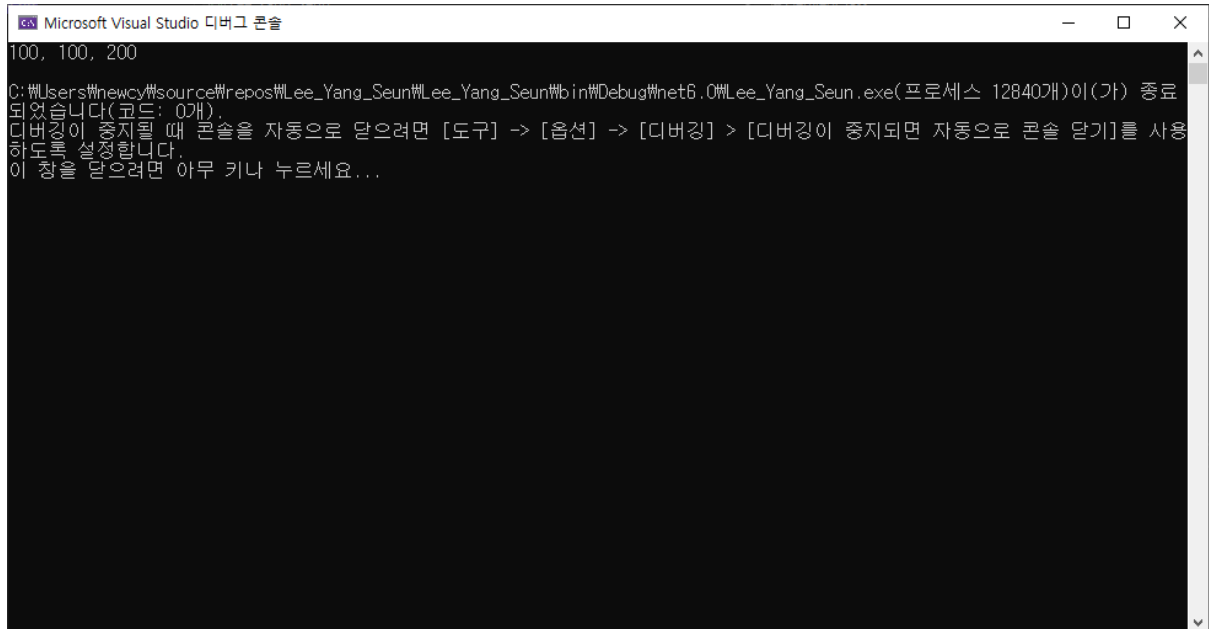
(2)

```
using System;

//5.6 (2)
참조 5개
class BaseClass
{
    참조 4개
    public bool Equal(object obj)
    {
        if (GetType() == obj.GetType()) return true;
        else return false;
    }
}
참조 4개
class DerivedClass : BaseClass { }
참조 0개
class Exercise5_6_2
{
    참조 0개
    public static void Main()
    {
        BaseClass b1 = new BaseClass();
        BaseClass b2 = new BaseClass();
        DerivedClass d1 = new DerivedClass();
        DerivedClass d2 = new DerivedClass();
        if (b1.Equal(d1))
            Console.WriteLine("Derived equals Base.");
        if (d1.Equal(b1))
            Console.WriteLine("Base equals Derived.");
        if (b1.Equal(b2))
            Console.WriteLine("Base equals Base.");
        if (d1.Equal(d2))
            Console.WriteLine("Derived equals Derived.");
    }
}
```

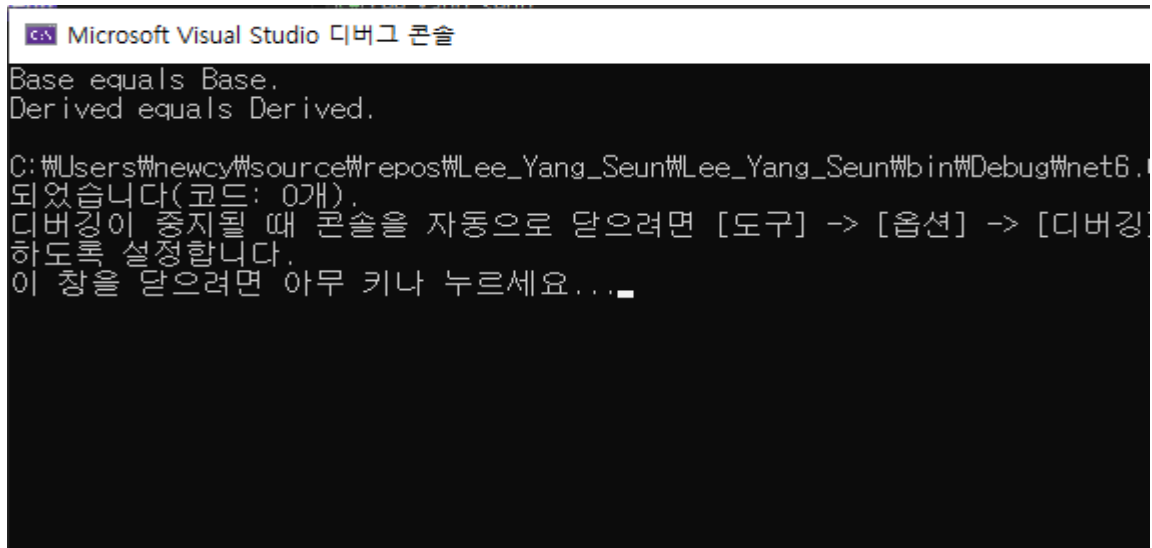
<결과>

(1)



```
Microsoft Visual Studio 디버그 콘솔
100, 100, 200
C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe (프로세스 12840개)이(가) 종료
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

(2)



```
Microsoft Visual Studio 디버그 콘솔
Base equals Base.
Derived equals Derived.
C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe (프로세스 12840개)이(가) 종
료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사
용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

<설명>

- (1)번 문제는 상속을 통해서 파생 클래스가 베이스 클래스의 변수를 상속받는 결과를 보여준다. 베이스 클래스의 변수 `supNum = 100` 이고, 파생 클래스의 변수 `subNum = 200` 이다. `Main()`에서 첫번째 줄은 베이스 클래스의 `supNum` 변수와 파생 클래스의 `supNum` 변수를 호출하고 있다. 베이스 클래스 안에는 `supNum` 변수에 대한 정보가 있으나, 파생 클래스 안에는 `supNum` 변수의 대한 정보가 없어 상위 클래스인 베이스 클래스로 올라

간다. 이로 인해, supNum 변수를 찾게 되고 두 변수의 값인 100을 출력한다. 두번째 줄은 파생 클래스의 subNum 변수를 호출하고 있다. 파생 클래스 안에는 subNum 변수의 정보가 있으므로 200을 출력한다.

- (2)번 문제는 베이스 클래스와 파생 클래스의 타입이 같은지를 보여준다. 베이스 클래스는 Equal 메소드가 존재하는데, 이 메소드는 인자 객체의 타입과 객체 자신의 타입이 같으면 True, 다르면 false를 반환한다. Main()에서 베이스 클래스 b1과 b2를 선언했고, 파생 클래스 d1과 d2를 선언했다. 4종류의 if문에서는 위에서 선언된 4개의 객체의 타입이 같은지의 유무를 판단한다.
- 첫번째 if문은 b1과 d1의 타입을 판단한다. b1의 타입은 BaseClass이고 d1의 타입은 DerivedClass이므로 두 객체의 타입은 같지 않으므로 false, 출력되는 내용이 없다.
- 두번째 if문은 d1과 b1의 타입을 판단한다. 이의 결과는 위와 마찬가지로 두 객체의 타입이 같지 않으므로 false, 출력되는 내용이 없다.
- 세번째 if문은 b1과 b2의 타입을 판단한다. b1의 타입은 BaseClass이고 b2의 타입은 BaseClass이므로 두 객체의 타입은 같다. true이므로 if문 속 출력문인 "Base equals Base." 문장이 출력된다.
- 네번째 if문은 d1과 d2의 타입을 판단한다. d1의 타입은 DerivedClass이고 d2의 타입은 DerivedClass이므로 두 객체의 타입은 같다. true이므로 if문 속 출력문인 "Derived equals Derived." 문장이 출력된다.

<5.6>

다음과 같은 추상 클래스를 이용하여 사각형과 원에 대한 클래스를 각각 정의하고 테스트하는 프로그램을 작성하시오.

```
using System;
참조 2개
abstract class Figure
{
    참조 4개
    public abstract void Area();
    참조 4개
    public abstract void Grith();
    참조 2개
    public abstract void Draw();
}
```

```

class Circle : Figure
{
    int r;
    double area;
    double grith;
    참조 0개
    public Circle()
    {
        this.r = 1;
    }
    참조 1개
    public Circle(int n)
    {
        this.r = n;
    }
    참조 2개
    public override void Area()
    {
        this.area = r * r * 3.14;
        Console.WriteLine("This circle's area is " + area);
    }
    참조 2개
    public override void Grith()
    {
        this.grith = r * 2 * 3.14;
        Console.WriteLine("This circle's grith is " + grith);
    }
    참조 1개
    public override void Draw()
    {
        throw new NotImplementedException();
    }
}

```

```

class Rect : Figure
{
    int w, h;
    double area;
    double grith;
    참조 0개
    public Rect()
    {
        this.w = 1;
        this.h = 1;
    }
    참조 1개
    public Rect(int n = 1, int m = 1)
    {
        this.w = n;
        this.h = m;
    }
    참조 2개
    public override void Area()
    {
        this.area = w * h;
        Console.WriteLine("This rectangle's area is " + area);
    }
    참조 2개
    public override void Grith()
    {
        this.grith = (w + h) * 2;
        Console.WriteLine("This rectangle's grith is " + grith);
    }
    참조 1개
    public override void Draw()
    {
        throw new NotImplementedException();
    }
}

```

```

class AbstractClasApp
{
    참조 0개
    public static void Main()
    {
        Circle obj1 = new Circle(100);
        Rect obj2 = new Rect(50, 80);
        obj1.Area();
        obj1.Grith();
        Console.WriteLine("");
        obj2.Area();
        obj2.Grith();
    }
}

```

-아래 코드는 MFC 코드입니다.

```
void CLEEREPORTView::OnDraw(CDC* pDC)
{
    CLEEREPORTDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 여기에 원시 데이터에 대한 그리기 코드를 추가합니다.
    CBrush brush1(RGB(0, 0, 150));
    CBrush* pOldBrush1 = pDC->SelectObject(&brush1);
    pDC->Rectangle(100, 100, 150, 180);
    brush1.DeleteObject();

    CBrush brush2(RGB(50, 50, 50));
    CBrush* pOldBrush2 = pDC->SelectObject(&brush2);
    pDC->Ellipse(300, 100, 500, 300);
    brush2.DeleteObject();
}
```

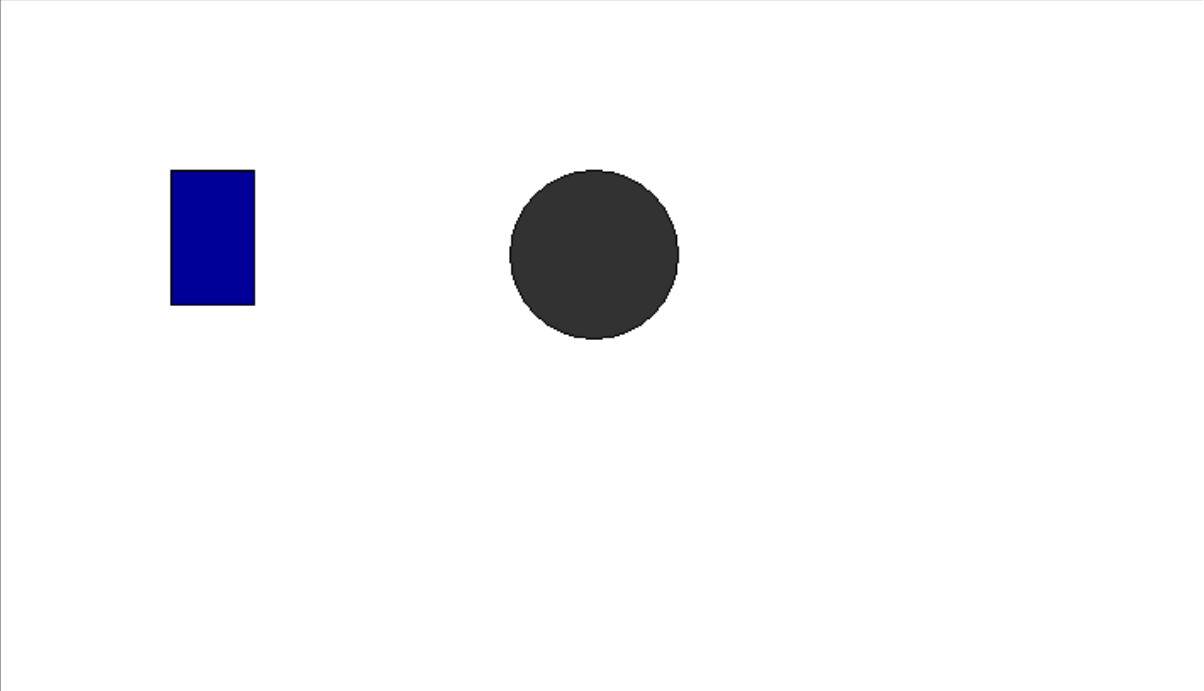
<결과>

Microsoft Visual Studio 디버깅 콘솔

```
This circle's area is 31400
This circle's grith is 628

This rectangle's area is 4000
This rectangle's grith is 260

C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



<설명>

- 추상 클래스 Figure에는 3가지 추상 메소드인 넓이를 구하는 메소드 `abstract void Area()`, 둘레를 구하는 메소드 `abstract void Girth()`, 도형을 그리는 메소드 `abstract void Draw()`가 있다. 추상 클래스 Figure를 상속받는 두 클래스 Rect와 Circle은 위의 추상 메소드를 재정의하여 사용한다.
- Circle클래스는 반지름 값 `r`, 넓이 `area`, 둘레 `grith` 등 3가지 변수가 있고, 매개변수가 없는 생성자의 경우 반지름의 값을 1로, 매개변수가 `n`인 생성자의 경우 반지름을 `n`으로 하여 생성한다. `override`를 사용하여 추상 메소드를 재정의 한 `Area()` 메소드는 원의 넓이를 구하는 공식인 $(\text{반지름} * \text{반지름} * \pi)$ 의 과정을 실행시켜 넓이를 출력한다. 마찬가지로, `override`를 사용하여 재정의 한 `Grith()` 메소드는 원의 둘레를 구하는 공식인 $(\text{반지름} * 2 * \pi)$ 의 과정을 실행시켜 둘레를 출력한다.
- Rect클래스는 밑변 `w`, 높이 `h`, 넓이 `area`, 둘레 `grith` 등 4가지 변수가 있고, 매개변수가 없는 생성자는 밑변과 높이가 1로, 매개변수가 `n`과 `m`인 생성자는 밑변이 `n`, 높이가 `m`으로 하여 생성한다. `override`를 사용하여 추상 메소드를 재정의 한 `Area()` 메소드는 사각형의 넓이를 구하는 공식인 $(\text{밑변} * \text{높이})$ 의 과정을 실행시켜 넓이를 출력한다. 마찬가지로, `override`를 사용하여 재정의한 `Grith()` 메소드는 사각형의 둘레를 구하는 공식인 $(\text{밑변} + \text{높이}) * 2$ 의 과정을 실행시켜 둘레를 출력한다.

- Main()에서는 반지름의 길이가 100인 Circle 객체 obj1과, 밑변이 50, 높이가 80인 Rect 객체 obj2를 생성하였고, 각각, Area()메소드와 Grith()메소드를 실행시켜 넓이와 둘레를 출력하도록 하였다.
- MFC를 실행시켜 도형이 그려지도록 한 코드는 RGB 값이 (0, 0, 150)의 색상을 갖는 사각형을 x좌표 100, y좌표 100 부터 시작하여 밑변이 50, 높이가 80인 사각형이 그려지도록 하였고, RGB 값이 (50, 50, 50)의 색상을 갖는 원을 x좌표 300, y좌표 100 부터 시작하여 반지름이 100인 원을 생성하도록 하였다.

<5.9>

다음 인터페이스를 보고 물음에 답하시오.

```
using System;

참조 34개
interface IOperation
{
    참조 10개
    void Insert(string str);
    참조 4개
    string Delete();
    참조 6개
    bool Search(string str);
    참조 6개
    string GetCurrentElt();
    참조 6개
    int NumOfElements();
}

참조 4개
class Stack : IOperation
{
    private string[] stack;
    int sp = -1;

    참조 0개
    public Stack()
    {
        stack = new string[100];
    }
    참조 1개
    public Stack(int size)
    {
        stack = new string[size];
    }
    참조 9개
    void IOperation.Insert(string str)
    {
        sp = sp + 1;
        stack[sp] = str;
    }
}
```



```
string IOperation.Delete()
```

```
{  
    sp--;  
    return stack[sp];  
}
```

참조 5개

```
bool IOperation.Search(string str)
```

```
{  
    int tmp = 0;  
    for(int i = 0; i <= sp; i++)  
    {  
        if (stack[i] == str)  
        {  
            tmp = 1;  
            break;  
        }  
    }  
    if(tmp == 0)  
    {  
        return false;  
    }  
    else  
    {  
        return true;  
    }  
}
```

참조 5개

```
string IOperation.GetCurrentElt()
```

```
{  
    return stack[sp];  
}
```

참조 5개

```
int IOperation.NumOfElements()
```

```
{  
    return sp+1;  
}
```

```

class Queue : IOperation
{
    int Front = 0, Rear = -1;
    int Count = 0;
    private string[] queue;

    참조 0개
    public Queue()
    {
        queue = new string[100];
    }

    참조 1개
    public Queue(int size)
    {
        queue = new string[size];
    }

    참조 9개
    void IOperation.Insert(string str)
    {
        if (Count <= queue.Length)
        {
            Rear = (Rear + 1) % queue.Length;
            queue[Rear] = str;
            Count++;
        }
    }

    참조 3개
    string IOperation.Delete()
    {
        string result = "";
        if (Count > 0)
        {
            result = queue[Front];
            Front = (Front + 1) % queue.Length;
            Count--;
        }
    }
}

```

```
    }  
    return result;  
}
```

참조 5개

```
bool IOperation.Search(string str)  
{  
    int tmp;  
    int check = 0;  
    bool result = false;  
  
    if(Count > 0)  
    {  
        if (Front <= Rear)  
        {  
            for (tmp = Front; tmp <= Rear; tmp++)  
            {  
                if (queue[tmp] == str)  
                {  
                    result = true;  
                    break;  
                }  
            }  
        }  
        else if (Front > Rear)  
        {  
            for(tmp = 0; tmp <= Rear; tmp++)  
            {  
                if (queue[tmp] == str)  
                {  
                    result = true;  
                    check++;  
                    break;  
                }  
            }  
            if (check == 0)  
            {  
                for (tmp = Front; tmp < queue.Length; tmp++)  
                {  
                    if (queue[tmp] == str)
```

```

        if (queue[tmp] == str)
        {
            result = true;
            break;
        }
    }
}

return result;
}

```

참조 5개

```

string IOperation.GetCurrentElt()
{
    return queue[Front];
}

```

참조 5개

```

int IOperation.NumOfElements()
{
    return Count;
}

```

참조 0개

```

class IOperationApp
{

```

참조 0개

```

    public static void Main()
    {
        Stack stk = new Stack(10);
        Queue que = new Queue(10);

        ((IOperation)stk).Insert("안녕하세요");
        ((IOperation)stk).Insert("저는 스택입니당");
        ((IOperation)stk).Insert("닷넷프로그래밍");
        ((IOperation)stk).Insert("이 문장은 지워지는 문장입니다");
    }
}

```

```

Console.WriteLine("현재 스택 탑에 있는 원소 : " + ((IOperation)stk).GetCurrentElt());
Console.WriteLine("스택에 존재하는 원소의 개수 : " + ((IOperation)stk).NumOfElements());
Console.WriteLine("이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : " + ((IOperation)stk).Search("이 문장은 지워지는 문장입니다"));

((IOperation)stk).Delete();
Console.WriteLine("");

Console.WriteLine("현재 스택 탑에 있는 원소 : " + ((IOperation)stk).GetCurrentElt());
Console.WriteLine("스택에 존재하는 원소의 개수 : " + ((IOperation)stk).NumOfElements());
Console.WriteLine("이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : " + ((IOperation)stk).Search("이 문장은 지워지는 문장입니다"));
//-----//
Console.WriteLine("");

((IOperation)que).Insert("이 문장은 지워지는 문장입니다");
((IOperation)que).Insert("안녕하세요");
((IOperation)que).Insert("저는 원형큐입니다");
((IOperation)que).Insert("닷넷프로그래밍");

Console.WriteLine("현재 큐의 Front에 있는 원소 : " + ((IOperation)que).GetCurrentElt());
Console.WriteLine("큐에 존재하는 원소의 개수 : " + ((IOperation)que).NumOfElements());
Console.WriteLine("이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : " + ((IOperation)que).Search("이 문장은 지워지는 문장입니다"));

((IOperation)que).Delete();
Console.WriteLine("");

Console.WriteLine("현재 큐의 Front에 있는 원소 : " + ((IOperation)que).GetCurrentElt());
Console.WriteLine("큐에 존재하는 원소의 개수 : " + ((IOperation)que).NumOfElements());
Console.WriteLine("이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : " + ((IOperation)que).Search("이 문장은 지워지는 문장입니다"));

```

< 결과 >



```

Microsoft Visual Studio 디버그 콘솔
현재 스택 탑에 있는 원소 : 이 문장은 지워지는 문장입니다
스택에 존재하는 원소의 개수 : 4
이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : True

현재 스택 탑에 있는 원소 : 닷넷프로그래밍
스택에 존재하는 원소의 개수 : 3
이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : False

현재 큐의 Front에 있는 원소 : 이 문장은 지워지는 문장입니다
큐에 존재하는 원소의 개수 : 4
이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : True

현재 큐의 Front에 있는 원소 : 안녕하세요
큐에 존재하는 원소의 개수 : 3
이 문장은 지워지는 문장입니다 - 존재하는 문장인가? : False

C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe(프로세스 9000개)이(가) 종료되었습니다(코드: 0x0).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

<설명>

- 클래스 Stack와 Queue는 인터페이스 IOperation을 상속받아 구현되었다.
- Stack 클래스는 매개변수 없는 생성자와 있는 생성자로 나뉘는데, 없을 경우 스택의 크기가 100으로, 있을 경우 해당 크기만큼 설정하여 생성되도록 하였다. Stack 클래스는 string 데이터들을 담을 stack 배열과 스택의 포인터인 sp, 인터페이스 IOperation에서 상속된 메소드들로 구성되어 있다. 앞으로 언급하는 메소드들은 인터페이스 IOperation에서 상속된 메소드들을 구현한 것이다.
- Insert() 메소드는 sp를 1증가시킨 후 매개변수인 string 데이터를 stack[sp]에 저장한다. Delete() 메소드는 현재 sp에 저장되어 있는 string 데이터를 반환한 후 sp를 -1 시킨다. Search() 메소드는 stack 배열에 처음부터 끝까지 살펴보며 인자로 받은 string 데이터와 일치하는 내용이 stack 배열 속에 있을 때 tmp를 1로 하여 tmp가 1일 때는 true를, 0일 때는 false를 반환한다. GetCurrentElt() 메소드는 현재 sp가 가리키는 stack의 데이터를 반환한다. NumOfElements() 메소드는 stack 배열에 존재하는 원소의 개수를 반환하는데, sp가 -1부터 시작하였으므로 1을 추가하여 sp를 반환한다.
- Queue 클래스는 string 데이터를 담을 배열 queue와 queue의 앞을 가리키는 포인터 Front, 끝부분을 가리키는 포인터 Rear, 배열의 데이터의 수를 나타내는 Count와 인터페이스 IOperation에서 상속된 메소드들로 구성되어 있다. Queue 클래스 생성자는 매개변수 없는 생성자와 있는 생성자로 나뉘는데, 없는 생성자의 경우 queue 배열의 크기를 100으로, 있는 생성자의 경우는 해당 매개변수만큼의 크기로 배열의 크기를 생성하여 객체를 생성한다.
- Insert() 메소드는 배열 속 데이터의 크기가 배열 전체 길이보다 작거나 같다면 Rear에 1을 더한 후 queue의 전체 길이만큼 나머지 연산자를 실행시켜 나온 값을 Rear에 저장시킨다. 그 후, queue[Rear]에 인자로 받은 string 데이터를 저장시킨 후 Count를 1증가시킨다. 위와 같이 나머지 연산자를 사용하여 Rear을 초기화 시킨 이유는 해당 큐를 원형 배열로 구현했기 때문이다. 원형 배열로 구현할 경우 지워진 데이터들의 공간을 재활용할 수 있기에 이런 식으로 구현하였다. 앞으로 설명할 메소드들도 원형 배열이라는 전제 하에 구현한 것이다. Delete() 메소드는 Count가 0보다 클 때, 현재 Front가 가리키는 queue 배열의 데이터를 잠시 result에 담은 후, Front를 1증가시킨 후, queue 배열의 길이만큼 나머지 연산자를 적용 후 Front를 초기화 한다. 그 후, Count를 -1 시키고 result를 반환한다.
- Search() 메소드의 경우는 조금 특이하다. 해당 Queue 클래스를 원형 배열로 구현했기 때문에 Front와 Rear의 크기가 엇갈리는 경우가 있다. 두가지의 경우를 대비하여 Front가 Rear보다 작거나 같을 경우, Front가 Rear보다 클 경우에 따라 다르게 작동하도록 구현하였다. 두 경우가 다르다 하나, 메소드의 작동원리는 같다. 인자로 받은 string 데이터와 같

은 데이터가 queue 배열 속에 있을 경우 bool 변수인 result를 true로 변환하여 result를 반환하는 원리이다. 만약 같은 데이터가 없다면 result의 원래 값인 false를 반환한다.

- GetCurrentElt() 메소드는 현재 Front가 queue 배열에서 가리키고 있는 데이터를 반환하는 메소드이고, NumOfElements() 메소드는 배열에 저장된 데이터의 개수인 Count를 반환하는 메소드이다.
- Main()에서는 크기가 10인 Stack 클래스 객체 stk와 Queue 클래스 객체 que를 생성한 후, 4가지 문장을 Insert() 메소드로 입력하였다. 그리고 각 객체에 GetCurrentElt() 메소드, NumOfElements() 메소드를 실행시켜 각각 객체에 탑과 Front에 있는 원소를 반환하여 출력하였고, stack 배열과 queue 배열의 원소의 개수를 반환하여 출력하였다. 그 후, Search() 메소드에 “이 문장은 지워지는 문장입니다”라는 내용의 문장의 유무 여부를 출력하고, Delete() 메소드를 작동시켜 각 객체에서 탑과 Front가 가리키는 데이터를 지웠다. 그 후에 다시 GetCurrentElt() 메소드, NumOfElements() 메소드를 실행시켜 탑과 Front가 가리키는 원소가 달라지는 것을 보였고, stack 배열과 queue 배열에 존재하는 원소의 개수가 달라지는 것도 보였다. Search() 메소드를 다시 실행시켜 “이 문장은 지워지는 문장입니다”의 내용을 담은 데이터가 있는지 확인해보면 True에서 False로 바뀌어 출력되는 모습을 볼 수 있다.

<5.11>

다음과 같은 인터페이스를 이용하여 사각형과 원에 대한 클래스를 각각 정의하고 테스트하는 프로그램을 작성하시오.

```
using System;

참조 12개
public interface IFigure
{
    참조 4개
    void Area();
    참조 4개
    void Girth();
    참조 2개
    void Draw();
}
```

```

class Rectangle : IFigure
{
    public int w, h;
    public double area;
    public double grith;

    참조 0개
    public Rectangle()
    {
        w = 1;
        h = 1;
    }

    참조 1개
    public Rectangle(int w, int h)
    {
        this.w = w;
        this.h = h;
    }

    참조 3개
    void IFigure.Area()
    {
        area = w * h;
    }

    참조 3개
    void IFigure.Girth()
    {
        grith = (w + h) * 2;
    }

    참조 1개
    void IFigure.Draw() { }    //MFC로 작성
}

```



```

class Circle : IFigure
{
    public int r;
    public double area;
    public double grith;

    참조 0개
    public Circle()
    {
        r = 10;
    }
    참조 1개
    public Circle(int r)
    {
        this.r = r;
    }

    참조 3개
    void IFigure.Area()
    {
        area = r * r * 3.14;
    }

    참조 3개
    void IFigure.Girth()
    {
        grith = r * 2 * 3.14;
    }

    참조 1개
    void IFigure.Draw() { }           //MFC로 작성
}

```

```

class IFigureApp
{
    참조 0개
    public static void Main()
    {
        Rectangle rect = new Rectangle(50, 80);
        Circle cir = new Circle(100);

        ((IFigure)rect).Area();
        ((IFigure)rect).Girth();
        Console.WriteLine("rect's area = " + rect.area);
        Console.WriteLine("rect's grith = " + rect.grith);

        Console.WriteLine("");

        ((IFigure)cir).Area();
        ((IFigure)cir).Girth();
        Console.WriteLine("cir's area = " + cir.area);
        Console.WriteLine("cir's grith = " + cir.grith);
    }
}

```

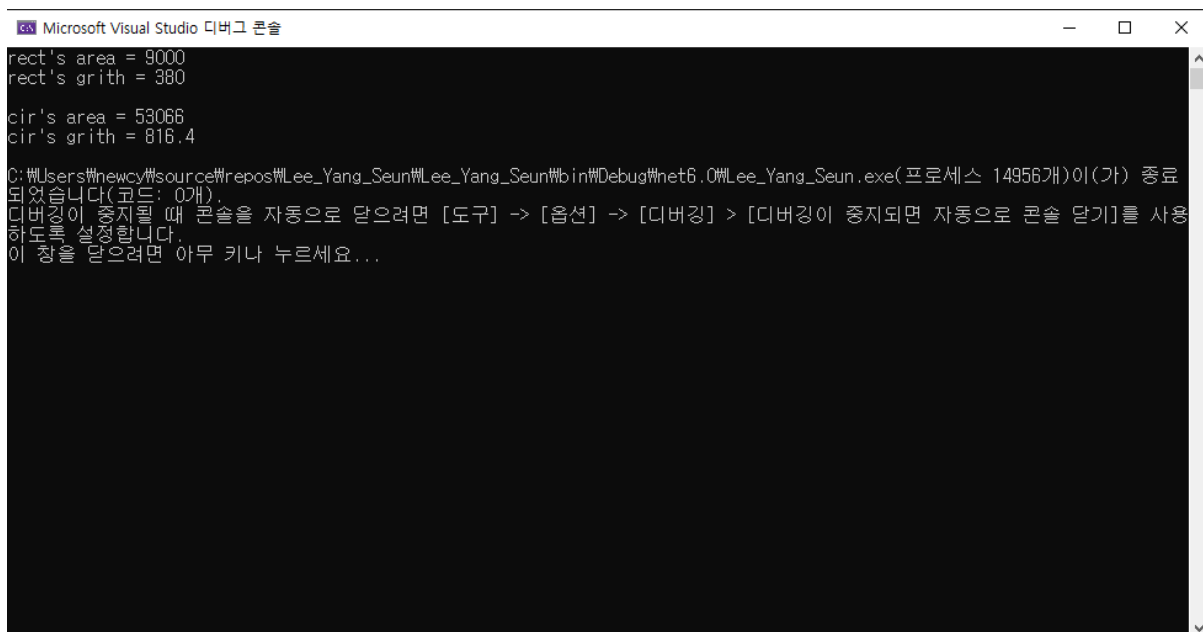
-MFC

```
void CLEEREPORTView::OnDraw(CDC* pDC)
{
    CLEEREPORTDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    // TODO: 여기에 원시 데이터에 대한 그리기 코드를 추가합니다.
    CBrush brush1(RGB(0, 0, 150));
    CBrush* pOldBrush1 = pDC->SelectObject(&brush1);
    pDC->Rectangle(100, 100, 150, 180);
    brush1.DeleteObject();

    CBrush brush2(RGB(50, 50, 50));
    CBrush* pOldBrush2 = pDC->SelectObject(&brush2);
    pDC->Ellipse(300, 100, 560, 360);
    brush2.DeleteObject();
}
```

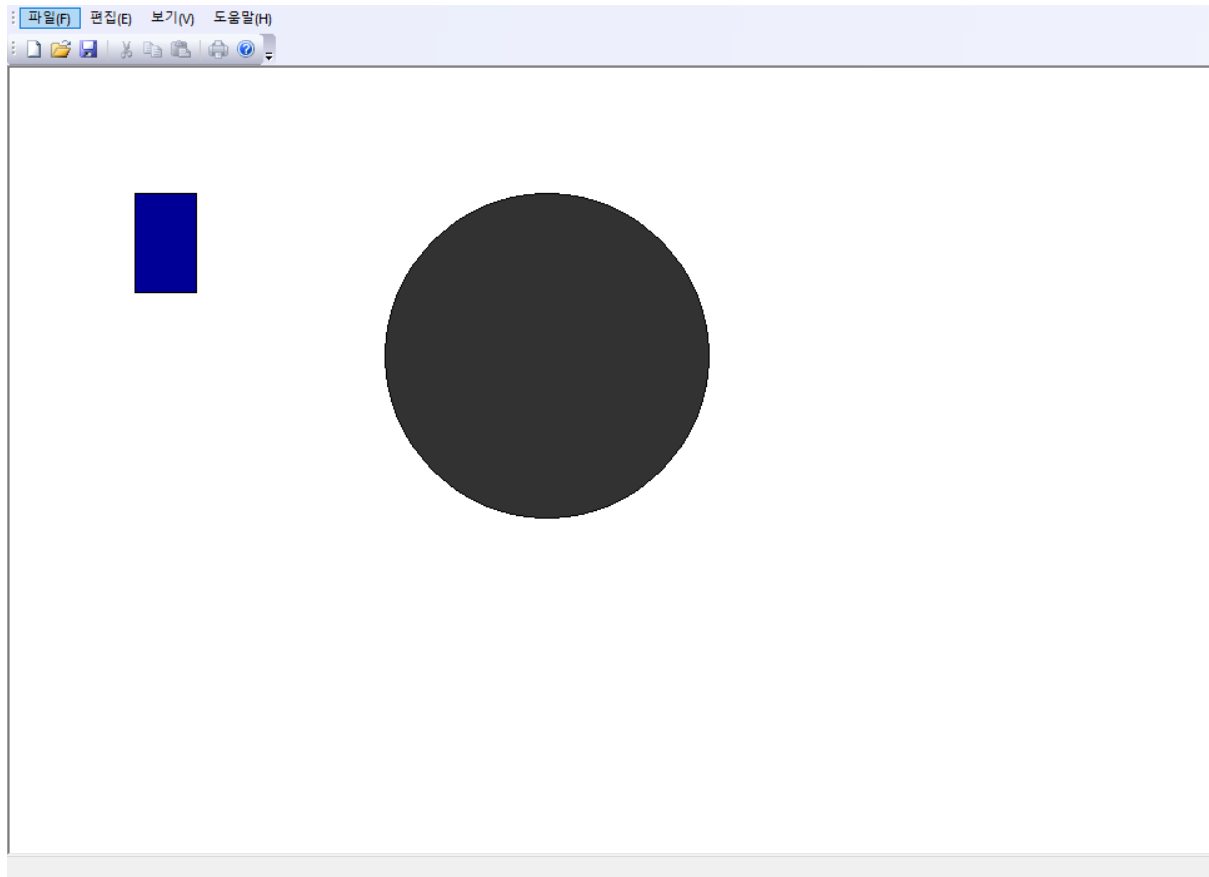
<결과>



```
Microsoft Visual Studio 디버그 콘솔
rect's area = 9000
rect's grith = 380

cir's area = 53066
cir's grith = 816.4

C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe(프로세스 14956개)이(가) 종료
되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



<설명>

- 인터페이스 IFigure은 넓이를 구하는 메소드 void Area(), 둘레를 구하는 메소드 void Girth(), 도형을 그리는 메소드 void Draw()로 구성되어 있다(Draw() 메소드는 MFC로 구현하였다.).
- 인터페이스 IFigure를 상속받은 클래스 Rectangle은 밑변과 높이를 나타내는 w와 h, 넓이를 나타내는 area, 둘레를 나타내는 girth의 변수들과, 매개변수 없는 생성자, 매개변수 있는 생성자, IFigure에서 상속받은 Area(), Girth(), Draw() 등의 메소드로 구성 되어있다.
- 매개변수가 없을 경우, $w = 1$, $h = 1$ 로 설정하여 생성되고, 있는 경우에는 해당 값으로 설정되어 생성된다. Area() 메소드는 사각형의 넓이 공식인 밑변(w) * 높이(h)로 변수 area를 초기화 시킨다. Girth() 메소드는 사각형의 둘레 공식인 (밑변(w) + 높이(h)) * 2로 변수 girth를 초기화 시킨다.
- 클래스 Rectangle과 마찬가지로 인터페이스 IFigure를 상속받은 클래스 Circle은 반지름을 나타내는 r과 넓이를 나타내는 area, 둘레를 나타내는 girth의 변수들과, 매개변수 없는 생성자, 매개변수 있는 생성자, IFigure에서 상속받은 Area(), Girth(), Draw() 등의 메소드로 구성 되어있다.

- 매개변수가 없을 경우, $r = 10$ 으로 설정하여 생성되고, 있는 경우에는 해당 값으로 설정되어 생성된다. Area() 메소드는 원의 넓이 공식인 반지름(r) * 반지름(r) * pi(3.14) 값으로 변수 area를 초기화 시킨다. Girth() 메소드는 원의 둘레 공식인 반지름(r) * 2 * pi(3.14) 값으로 변수 girth를 초기화 시킨다.
- Main() 에서는 Rectangle 클래스의 객체 rect와 Circle 클래스의 객체 cir을 생성하였다. rect 객체는 밑변이 50, 높이가 80으로 설정되었고, cir 객체는 반지름이 130으로 설정되었다. 각 객체에 Area() 메소드와 Girth() 메소드를 실행시켜 area 변수와 girth 변수를 초기화 시킨 후 WriteLine을 실행시켜 각 객체의 넓이와 둘레를 출력하였다.
- MFC 상에서 Rectangle 클래스와 Circle 클래스를 나타내기 위해 사각형은 x좌표 100에서 밑변 크기인 50만큼, y좌표 100에서 높이 크기인 80만큼의 크기로 설정하여 출력하였고, 원은 x좌표 300, y좌표 100에서 시작하여 반지름의 크기인 130만큼 크기를 설정하여 x좌표 560, y좌표 360인 원을 출력하였다.