

## ■ 수업활동일지: 플립러닝(개별 제출)

교과목명	비주얼프로그래밍	분반	002
제출일자	2022년 10월 19일	교수자명	장성진
이름	김민오	학번	20193081
팀명			

### ■ 수업내용

#### <3장: 실습문제(p149)>

3-9. (7)-1번 소스코드

```

// 3-9. (7)-1번 문제
참조 0개
static void Main(string[] args)
{
    int a = 1, b = 1, sum = 0;

    for (int i = 1; i <= 10; i++)
    {
        a = a * b; //제일 처음 값을 만들기 위해 a=1, b=1을 채택
        b++; //b를 증가 후 루프 종료
        sum = sum + a; //해당 증가분인 a는 sum에 더해짐 (괄호 느낌)
    }

    // factorial 덧셈 문제! 재귀로도 처리가 가능할지도?
    Console.WriteLine("결과값:{0}", sum);
}

```

#### (7)-1번 결과화면

```

C:\WINDOWS\system32\cmd.exe
결과값:4037913
계속하려면 아무 키나 누르십시오 . . .

```

## (7)-2번 소스코드

```

참조 0개
static void Main(string[] args)
{
    double mol = 1, demol = 1, sum = 0; //mol은 분자, demol은 분모이다.

    Console.WriteLine("S = "); //결과값 출력 형태 : (S=)
    for (int i = 1; i <= 10; i++)
    {
        if (i % 2 == 0) //해당 분수에서는 1에서부터 시작하므로 짝수번째 순서일때 실제 연산은 - 연산을 하고 표기는 +로 한다.
        {
            Console.WriteLine("{0}/{1} +", mol, demol); //후위 표기는 + 표기
            sum = sum - (mol / demol); //전위 실제 계산은 - 연산
        }

        else if (i % 2 == 1) // 홀수번째 순서 일 때, 실제 연산은 + 연산을 하고 표기는 - 로 한다.
        {
            Console.WriteLine("{0}/{1} -", mol, demol); //- 표기
            sum = sum + (mol / demol); //+ 연산
        }

        demol++; //분수가 1씩 증가하므로 루프가 종료될때마다 ++ 연산을 해준다.
    }
    Console.WriteLine("");
    Console.WriteLine("결과값:{0}", sum); //최종 결과 값 표기
}

```

## (7)-2번 결과화면

```

C:\WINDOWS\system32\cmd.exe
S = (1/1) -(1/2) +(1/3) -(1/4) +(1/5) -(1/6) +(1/7) -(1/8) +(1/9) -(1/10) +
결과값:0.645634920634921
계속하려면 아무 키나 누르십시오 . . .

```

## 3-9. (8)번 소스코드

```

참조 0개
static void Main(string[] args)
{
    int start = 2, end = 5; //시작을 2단으로 시작, 단위는 4개의 단부터 끝 (끝단 -1 만큼!)

    while (end <= 10) //목표 단수가 9단이므로 (만약 더 많은 결과를 출력하고싶을 시 해당 부분의 숫자만을 조절하면된다.)
    {
        for (int i = 1; i <= 9; i++) //2x1, 3x1... 처럼 열 단위로 출력
        {
            for (int j = start; j <= end; j++) // 시작 단을 start변수, 종료단을 end변수로 둬으로써 변수로 컨트롤하게끔 만들
            {
                Console.WriteLine("{0,2} x {1,2} = {2,2}, ", j, i, i * j); //포매팅 기술 : { 0, 2 } 는 2칸 공백 만드는 기술
            }
            Console.WriteLine(""); //열단위로 끊어내는 개행 (이것을 통해 세로 행 출력이 가능)
        }
        Console.WriteLine(""); // 해당 부분도 마찬가지로
        //해당 부분을 통해 4단씩 출력 가능(while문으로만 통제)
        start = end + 1;
        end = end + 4;
    }
}

```

### (8)번 결과화면

```
C:\WINDOWS\system32\cmd.exe

2 * 1 = 2, 3 * 1 = 3, 4 * 1 = 4, 5 * 1 = 5,
2 * 2 = 4, 3 * 2 = 6, 4 * 2 = 8, 5 * 2 = 10,
2 * 3 = 6, 3 * 3 = 9, 4 * 3 = 12, 5 * 3 = 15,
2 * 4 = 8, 3 * 4 = 12, 4 * 4 = 16, 5 * 4 = 20,
2 * 5 = 10, 3 * 5 = 15, 4 * 5 = 20, 5 * 5 = 25,
2 * 6 = 12, 3 * 6 = 18, 4 * 6 = 24, 5 * 6 = 30,
2 * 7 = 14, 3 * 7 = 21, 4 * 7 = 28, 5 * 7 = 35,
2 * 8 = 16, 3 * 8 = 24, 4 * 8 = 32, 5 * 8 = 40,
2 * 9 = 18, 3 * 9 = 27, 4 * 9 = 36, 5 * 9 = 45,

6 * 1 = 6, 7 * 1 = 7, 8 * 1 = 8, 9 * 1 = 9,
6 * 2 = 12, 7 * 2 = 14, 8 * 2 = 16, 9 * 2 = 18,
6 * 3 = 18, 7 * 3 = 21, 8 * 3 = 24, 9 * 3 = 27,
6 * 4 = 24, 7 * 4 = 28, 8 * 4 = 32, 9 * 4 = 36,
6 * 5 = 30, 7 * 5 = 35, 8 * 5 = 40, 9 * 5 = 45,
6 * 6 = 36, 7 * 6 = 42, 8 * 6 = 48, 9 * 6 = 54,
6 * 7 = 42, 7 * 7 = 49, 8 * 7 = 56, 9 * 7 = 63,
6 * 8 = 48, 7 * 8 = 56, 8 * 8 = 64, 9 * 8 = 72,
6 * 9 = 54, 7 * 9 = 63, 8 * 9 = 72, 9 * 9 = 81,

계속하려면 아무 키나 누르십시오 . . .
```

### 3-9. (10)번 소스코드

```
static void Main(string[] args)
{
    int thing = 974; //물건 가격
    int money = thing; //해당 물건 가격을 돈 계산 프로그램 알고리즘에 넣기 위해 money라는 변수에 대입

    //money 입력받기

    int ex_500 = money / 500; //최대금액에서 500원짜리 갯수
    money -= money % 500; //나머지는 남은 금액으로 이동

    int ex_100 = money / 100; //위에서 남은 나머지 금액에서 나오는 100원짜리 최대갯수
    money -= money % 100; //나머지는 남은 금액으로 이동

    int ex_50 = money / 50; //나머지 금액에서 나오는 50원짜리 최대 갯수
    money -= money % 50;

    int ex_10 = money / 10; //위와 같은 방법
    money -= money % 10;

    int ex_5 = money / 5; //위와 같은 방법
    money -= money % 5;

    int ex_1 = money; //남은 돈은 반드시 1원 단위이므로 그대로 대입

    Console.WriteLine("물건 가격 : {0}원", thing);
    Console.WriteLine("===최적의 거스름돈===");
    Console.WriteLine("500원 : {0}개, 100원 : {1}개, 50원 : {2}개, 10원 : {3}, 5원 : {4}, 1원 : {5}", ex_500, ex_100, ex_50, ex_10, ex_5, ex_1);
    //포맷팅을 통해 출력
}
```

### (10)번 결과화면

```
C:\WINDOWS\system32\cmd.exe

물건 가격 : 974원
===최적의 거스름돈===
500원 1개, 100원 4개, 50원 1개 10원 2개, 5원 0개 1원 4개
계속하려면 아무 키나 누르십시오 . . .
```

## <4장 : 이론문제>

4-1. 다음 괄호안에 적당한 용어를 쓰시오.

### | 연습문제 |

4-1 다음 괄호 안에 적당한 용어를 쓰시오.

- (1) 클래스는 객체의 구조와 (행위)를 정의하는 방법으로 C# 프로그램의 기본 단위이다.
- (2) 객체를 생성하는 방법은 (new) 연산자를 사용하여 생성하는 형태로 기술한다. → 생성자 생성!
- (3) 리터럴 (literal)을 사용하여 선언된 변수의 값이 변할 수 없다는 속성을 갖는다. → 유산변수들 (const static final?)
- (4) 필드에서, 접근 수정자가 생략된 경우는 (private)로 선언한 것과 동일하다. → my 필드(변수) final?
- (5) 메소드를 정의할 때 사용하는 매개변수를 (형식 매개변수)라 부르고 호출할 때 사용하는 매개변수를 (실 매개변수)라 부른다.
- (6) 매개변수 수정자(parameter modifier)에는 (ref)와 (out)이 있다. → 호환을 하거나 하지 않는 차이
- (7) 시그네처란 메소드를 구별하는 데 쓰이는 정보로 메소드의 이름, 매개변수의 개수, 그리고 매개변수의 형을 의미한다.
- (8) using 문에서, 자원(resource)의 의미는 클래스나 구조체를 말하며 (Dispose()) 메소드를 반드시 갖고 있어야 한다. (반수) p179 → 자원해제를 함
- (9) 프로퍼티란 의미는 (메소드)이지만 (필드)처럼 사용되는 클래스 멤버이다.
- (10) 프로퍼티는 값을 지정하는 (set)과 값을 참조하는 (get)으로 구성된다.
- (11) 셋-접근자에서 사용한 (value)라는 지정어는 마치 매개변수와 같은 역할을 하며 배경 연산의 우측식의 연산된 결과를 전달받게 된다. ⇒ set { } = value; → 매개변수처럼 전이됨
- (12) 셋-접근자 없이 게-접근자만이 정의되어 있는 프로퍼티를 (읽기 전용 프로퍼티)라 부른다. (반) 쓰기 전용 프로퍼티
- (13) 인덱서는 객체를 (배열)을 통해서 다룰 수 있는 방법이다. (readonly 프로퍼티) = readonly 프로퍼티 = write only 프로퍼티
- (14) 연산자 중복은 기존에 있던 연산자를 특정 클래스만을 위한 연산 의미를 가지도록 (재정의)하는 것이다. → public static extern [변환형] operator [연산자] (매개변수) { ... }
- (15) C#에서, 기본 자료형 중에 연산자 중복이 가능한 연산자의 개수는 (24)개이다. (자신의 클래스는 자신의 클래스만의 연산자를 정의할 수 있음)
- (16) 단항 ++, -- 연산자는 매개변수 자료형은 ( )이고 반환형은 ( )이어야 한다.
- (17) 연산자 중복 시 반드시 대칭적 방식으로 정의해야 하는 연산자는 (true)와 (false), (==)와 (!=), (<)와 (>), 그리고 (<=)와 (>=)가 있다. (비교연산자 중복은 대칭성으로 중복하여야 함)
- (18) 매리게이터는 주로 이벤트와 스레드에 사용된다. 따라서 이들을 위해 미리 정의된 델리게이트가 있는데 이벤트를 위한 ( )가 있다. 이벤트 핸들러
- (19) 매리게이터를 정의할 때, 연결된 메소드와 ( )가 정확히 일치해야 한다. (형태, 시그네처)

(ex) f. Navigator = {  
set { }  
get { }  
지정어로 지정!

재정의 불가능 연산자  
1. []  
2. ()  
3. &&  
4. ||  
5. 조건 실행 연산자 (?:)  
6. 배정 연산자 (=)  
→ (2, 3, 4)는 자료형 불변!

## 4-2. 다음 객관식 문제를 푸시오.

- ?
- (20) 엘리게이트를 클래스 밖에서 정의할 때, **public**으로 선언하면 다른 프로그램에서도 참조할 수 있으나 ( )로 선언하면 같은 프로그램 내에서만 참조할 수 있다.
  - (21) 리제이트에 메소드를 편집할 때 사용하는 편집자는 ( )이고 추가할 때 사용하는 편집자는 ( )이며 제거할 때 사용하는 편집자는 ( )이다.
  - (22) 이벤트란 사용자 ( )에 의해 발생하는 사건을 의미한다.
  - (23) 이벤트 수정자의 ( )와 ( )는 모두 메소드 수정자와 같다.
  - (24) 이벤트의 종류에는 크게 ( ) 이벤트와 ( ) 이벤트가 있다.

### 4.2 다음 객관식 문제를 푸시오.

- (1) 클래스를 통하여 얻을 수 있는 **객체지향 개념**은? → **유연성, 이식성, 재사용성 ↑**  
 ① 자료 추상화 (1) ② 상속성 클래스의 본질 → ADT 생성 (구상화!)  
 ③ 다형성 ④ 유지 보수성  
 ⑤ 정보 은닉화
- (2) 접근 수정자를 통하여 얻을 수 있는 장점은? **private**이 클래스의 기본 → **캡슐화 (=정보 보호!)**  
 ① 재사용성 ② 가독성  
 ③ 이식성 ④ 정보 은닉화  
 ⑤ 자료 추상화
- (3) 클래스에서 사용할 수 있는 수정자(modifier)는?  
 ① extern ② volatile  
 ③ readonly ④ virtual  
 ⑤ **static** **public** **static** **internal** **sealed** **abstract** **partial** **private** **protected**  
 — 봉인클래스 — 추상클래스
- (4) 필드에서 사용할 수 있는 수정자(modifier)는?  
 ① extern ② sealed  
 ③ **new** ④ virtual **public** **protected** **private** **protected** **internal** **new** **static** **readonly** **volatile**  
 ⑤ abstract
- (5) 접근 수정자를 명시하지 않으면 다음 중 어느 것과 동등한가?  
 ① **private** — 클래스의 기본 ③ **protected**  
 ② **internal** = **접근화** ④ **protected internal**  
 ⑤ **public**

- (6) 동일 네임스페이스뿐만 아니라 파생 클래스에서도 사용할 수 있는 접근 수정자는?  
 ① **public** — 공통 ② **protected internal** — **같은 클래스 + 파생 클래스에서만 사용 가능**  
 ③ **internal** — **같은 클래스 + 파생 클래스에서만 사용 가능** ④ **protected** — **같은 클래스 + 파생 클래스에서만 사용 가능**  
 ⑤ **private** — **같은 클래스에서만 사용 가능**
- (7) 상수 멤버(constant member)와 읽기전용 필드에 대한 설명이 틀린 것은?  
 ① 한번 값이 결정되면 모두 다 변경할 수 없다.  
 ② 상수 멤버는 컴파일 시간에 값이 결정되고 읽기전용 필드는 실행시간에 값이 결정된다.  
 ③ 상수 멤버의 자료형은 기본 자료형만 가능하고 읽기전용 필드의 자료형은 클래스형도 될 수 있다. → **상수 멤버의 자료형은 기본 자료형만 가능**  
 ④ 읽기전용 필드가 상수 멤버보다 프로그램 실행상 더 효율적이다. (효율성은 소일에 따라 다르다)  
 ⑤ **상수 멤버는 const 지정어를 읽기전용 필드는 readonly 수정자를 사용한다.**

- (8) 메소드에서 사용할 수 없는 수정자(modifier)는?  
 ① extern ② sealed  
 ③ **static** ④ **volatile** **new** **static** **override** **abstract** **virtual** **sealed** **extern**  
 ⑤ abstract
- (9) 다음 중 인스턴스 객체의 필드와 메소드를 참조하는 형태로 바른 것은? 3  
 ① **objectName** → **fileName** → **리소스 연산자는 사용 X**  
 ② **fileName** → **objectName**  
 ③ **objectName.fileName** → 예) **Class\_obj** objectName = new Class\_obj();  
 ④ **fileName.objectName** → 반대! (X)  
 ⑤ **ClassName.fileName** → **정적 필드일 때만 가능!** (static으로 선언된 class)

- (10) 메소드의 시그네처에 해당되지 않는 것은?  
 ① 매개변수의 개수 ② 매개변수의 형  
 ③ 메소드의 이름 ④ 메소드의 반환형  
 method signature ⇒ 메소드 구현 방식의 정보  
 1. 이름  
 2. 개수  
 3. 시그니처



(11) 생성자와 소멸자에 대한 설명이 바른 것은?

- (1) 생성자는 new에 의해 소멸자는 delete에 의해 호출된다. Dispose() [45] Finalize() [48] delete 명령어는 X
- ✓ 생성자의 이름과 소멸자의 이름은 같다.
- (3) 생성자와 소멸자는 모두 매개변수를 가질 수 있다. → 생성자에서 매개변수를 ~~정의~~ 선언할 수 있다.
- (4) 생성자는 반환형을 가질 수 있으나 소멸자는 가질 수 없다. → 반환형을 가질 수 X
- (5) 생성자와 소멸자는 모두 중복으로 정의될 수 있다. → 소멸자는 중복으로 정의될 수 X

(12) 멤버 수정자와 그에 해당하는 지정어 개수를 연결한 것이다. 틀린 것은?

- (1) 클래스 수정자 - 8개
  - (2) 필드 수정자 - 8개
  - ✓ (3) 접근 수정자 - 5개
  - (4) 메소드 수정자 - 11개
  - (5) 매개변수 수정자 - 2개
1. public 2. private 3. protected  
4. internal 5. protected internal 6. static  
L ref, out

(13) 필드 수정자에 관한 설명 중 틀린 것은?

- ✓ (1) readonly와 volatile은 함께 사용할 수 있다. volatile의 뜻 : 외부적인 요인으로 그 값이 언제든지 바뀔 수 있음.
  - (2) static과 readonly는 함께 사용할 수 있다.
  - (3) static과 volatile은 함께 사용할 수 있다.
  - (4) private와 static은 함께 사용할 수 있다.
  - (5) private와 readonly는 함께 사용할 수 있다.
- ↓  
static 전역변수에 공동 사용 →  
readonly는 읽기 전용으로 같이 바꿀 수 X  
→ volatile과 의미가 의역의 반대

(14) 정적 필드와 인스턴스 필드의 차이점은 무엇인가?

- (1) 정적 필드는 static으로 선언한 필드이며 생성된 객체를 통하여 참조한다.
- (2) 인스턴스 필드는 지정어 instance로 선언한 필드를 말하며 생성된 객체를 통하여 참조한다. instance 지정어 X → new 지정어
- ✗ (3) 정적 필드는 인스턴스 필드를 초기화하는데 사용할 수 없다. (X) 인스턴스 필드는 객체 존재해야 함
- (4) 인스턴스 필드는 클래스 이름이나 객체를 통하여 참조할 수 있다.
- ✓ (5) 정적 필드는 클래스 단위로 존재하고 인스턴스 필드는 객체 단위로 존재한다. 클래스 참조하여... 객체를 참조...

(15) 매개변수 수정자에 대한 설명 중 바른 것은?

- (1) 매개변수 수정자에는 ref, out, in 등이 있다.
- ✓ (2) ref와 out의 차이점은 초기화뿐이다. ref는 매개변수 전달시, 반드시 초기화된 상태에 → 아니면 error  
out은 초기화하지 않아도 됨!
- (3) 매개변수 수정자가 생략되면 참조 호출이 된다.
- (4) in은 값을 전달하기 위해서 out은 주소를 전달하기 위해 사용된다.
- (5) C#에서 참조 호출을 할 수 있는 방법은 오직 객체를 사용하는 것이다.  
+ (4) (ref, out) 매개변수 수정자 사용!

반드시 대응!

(16) C# 언어에서 Main() 메소드의 매개변수의 자료형은?

- 1 int
- 2 string
- 3 object

- 1 string
- 2 object

→ public static void Main(string[] args)  
⇒ String 배열을 매개변수로 가질 수 있다

(17) 다음 중 프로퍼티의 수정자가 될 수 없는 것은?

- 1 internal
- 2 override
- 3 extern

- 1 virtual
- 2 readonly

new  
static  
virtual  
sealed  
override  
abstract  
extern

X 메소드 수정자와 동일

(18) 다음 중 프로퍼티에 대한 설명으로 맞는 것은? 4

- 1 프로퍼티는 반드시 대응되는 private 필드가 존재해야 한다. → get, set 등 용도에 맞게 존재
- 2 프로퍼티의 의미는 필드이지만 참조는 메소드처럼 사용한다. (반대!)
- 3 프로퍼티는 반드시 get-접근자와 set-접근자를 갖고 있어야 한다. → 용도에 맞게
- 4 프로퍼티 수정자의 종류와 의미는 모두 메소드와 같다.
- 5 프로퍼티의 참조는 항상 배경문의 오른쪽에서만 가능하다. → get, set 등

(19) 다음 중 인덱서의 수정자로 사용할 수 없는 것은?

- 1 static
- 2 override
- 3 extern

- 1 virtual
- 2 internal

public  
protected  
internal  
private  
new

virtual  
override  
abstract  
sealed  
extern

특히 static 사용 불가!

(20) 다음 중 인덱서에 대한 설명으로 맞는 것은?

- 1 인덱서는 배열 연산자를 중복한 것이다. (프로퍼티와 동일)
- 2 인덱서는 상속이 불가능하다. (프로퍼티의 protected 문제!)
- 3 인덱서는 프로퍼티의 특별한 형태이다.
- 4 인덱서에서 매개변수의 형은 정수형만이 가능하다.
- 5 인덱서에서 다차원 배열 형태는 사용할 수 없다.

string도 가능 (여러개의 문자열)

✓ 3 안 [ ]로 가능

(21) 연산자 중복에 대한 설명이 맞는 것은? 4

- 1 연산자 중복은 ~~같은~~ 연산자에 대해 중복이 가능하다. (X)
- 2 연산자 +, -, \*, / 연산자는 매개변수의 반환형은 자료의 원래 형태와 동일 가능하다. (X)
- 3 특정 자료형의 형 변환은 연산자 중복이 불가능하다. (X)
- 4 true/false 연산자 중복의 반환형은 bool 형이다.
- 5 <=와 >=는 연산자 중복시 함께 정의되어야 한다.

<= 이다!

(22) 연산자 중복에 대한 설명 중 틀린 것은? 4

- (1) 연산자 중복은 연산자의 의미만을 세심하게 뿐 문법적 구조는 재정의할 수 없다.  
 (2) B.B. II 연산자는 재정의할 수 없다. (3) (=) 연산자 재정의 불가  
 (4) 매정 연산자(=)는 연산자 중복으로 재정의할 수 없다.  
 (5) 비교 연산자는 연산자 중복시 함께 정의하여야 한다. → !, ~ 같은 비연산자 해당 X  
 (6) 매개변수의 개수가 다함 연산자는 1개이고 이하 연산자는 2개이다.  
 (op1, op2)

(23) 연산자 중복에서 연산자에 따른 매개변수 형과 반환형에 대한 규칙이 바른 것은? 3

- ① ~ 연산자인 경우에, 매개변수 형은 자신의 클래스이고 반환형은 bool 형이어야 한다. (X) 반환형
- ② ++ 연산자인 경우에, 매개변수 형은 자신의 클래스이고 반환형은 파생 클래스 이어야 한다. (X) + 자식의 클래스
- ③ << 연산자인 경우에, 첫 번째 매개변수 형은 자신의 클래스이고 두 번째 매개변수 형은 int 형이어야 한다. (시프 연산자)
- ④ true/false 연산자인 경우에, 매개변수형도 반환형도 모두 bool 형이다.
- ⑤ + 연산자인 경우에, 두개의 매개변수형도 반환형도 자신의 클래스 형이어야 한다.

(24) 사용자 정의 형 변환(user-defined type conversion)에 관한 설명 중 맞는 것은?

- ① 형 변환 연산자의 이름은 특수 문자로 이루어져야 한다.  $Xtype Ytype$
  - ② 형 변환 연산자를 정의할 때, 형 변환 이외에 부가적인 작업을 할 수 있다.
  - ③ `implicit`로 선언된 형 변환 연산자는 캐스팅 연산자로 사용할 수 있다. *중요, 연산 5개!*
  - ④ 형 변환 연산자의 매개변수의 개수는 1개나 2개가 될 수 있다. *2개는 `explicit` 캐스팅 연산자로*
  - ⑤ 형 변환 연산자의 이름은 자료형을 나타내는 지정어이어야 한다. *implicit 목적(자동)*
- ④ 클래스 이름 1개  
⑤ 1개

(25) 델리게이트 정의에 관한 설명 중 맞는 것은?

- ㉠ 델리게이트의 수정자는 public과 internal만이 가능하다
- ㉡ 연결될 메소드와 반환형, 매개변수의 개수와 형이 정확히 일치해야 한다.
- ㉢ 매개변수를 갖는 델리게이트는 정의할 수 없다.
- ㉣ 이벤트를 위한 델리게이트를 정의할 때 지정어 delegate 대신에 event를 사용한다.
- ㉤ 델리게이트는 메소드 형태에 따라 반드시 정의해서 사용해야 한다.

4) 클래스와 구조체에 관한 설명 중 틀린 것은?

- 클래스는 메소드를 가질 수 있지만 구조체는 메소드를 가질 수 없다. — 메소드, 필드, 변수, 상인!
- 클래스는 참조형이지만 구조체는 값형이다.
- 클래스는 힙(heap)에 저장되고 구조체는 스택(stack)에 저장된다.
- 클래스는 상속이 가능하지만 구조체는 상속할 수 없다. — 장정래형 컷!
- 매개 변수의 경우 클래스는 참조가 복사되고 구조체는 내용이 복사된다.



#### 4-4. 다음 용어를 비교 설명하시오.

##### (1) public vs. private

- 둘 다 접근 수정자에 해당하며, public은 같은클래스, 파생클래스, 동일 네임스페이스, 다른 네임스페이스, 모든 클래스에서 사용가능한 반면 private은 오직 같은 클래스에서만 사용 가능하게 하는 접근수정자이다.

예) `private int private_field; public int public_field;`

##### (2) protected vs. protected internal

- 둘의 공통점은 접근수정자 중 파생클래스 관련한 접근수정자 이다. 단, protected는 같은 클래스와 파생클래스에서만 접근가능하게 한다면, protected internal은 같은 클래스, 파생클래스 뿐만 아니라 동일한 네임스페이스까지 접근 가능하게 하는 일종의 확장판의 개념으로 이해할 것.

##### (3) instance variable vs. static variable

-instance variable은 객체 변수, 인스턴스 필드라고 하며 객체가 생성되고 그 객체마다 가질 수 있는 필드를 의미한다. 해당 필드는 객체를 통해 참조할 수 있다. new 지정어로 객체를 생성될 때 해당 객체를 통해 필드에 접근할 수 있다. static variable은 정적 변수, 정적 필드라고 하며 static 수정자 통해 선언한 필드를 의미한다. 해당 필드는 (반드시 꼭) 클래스의 이름을 통해서만 가능하며 객체를 통해서서는 절대 참조할 수 없다. (추가로 인스턴스 객체에 정적 필드는 존재할 수 없다. 정적 필드는 객체 생성 전에 만들어지기 때문이다.)

##### (4) call by value vs. call by reference

- call by value = 값에 의한 호출을 의미하며 실 매개변수의 값이 복사되어서 형식 매개변수로 전달되는 것을 의미하며 이때 실제로 전달한 값은 값이 복사되어 전달되었기 때문에 변하지 않는다.

call by reference = 주소 호출(call by address)라고도 불리며, 실 매개변수의 주소가 형식 매개변수로 전달되는 것을 의미하며, 메소드를 호출할 시 실제 값이 변경되게 된다. 이 때 참조호출을 이루는 방법은 두 가지가 존재하는데 매개변수 수정자를 사용하는 방법(ref와 out)과 객체를 참조하여 바꾸는 방법이 존재한다.

##### (5) ref vs. out

- ref와 out 기능은 둘 다 주소를 전달해주는 기능으로 동일하지만, ref는 메소드로부터 매개변수가 전달될 때, 반드시 초기화된 값이 와야하지만, out은 그러지않아도 된다는 것이 유일한 차이점이다.

##### (6) property vs. method

- 프로퍼티는 클래스의 private 필드를 형식적으로 다루는 일종의 메소드이고, 메소드(method)는 객체의 행위를 기술하는 방법으로 상태를 검색, 변경하는 작업, 특정한 행동을 처리하는 코드를 포함한다.

메소드는 일종의 큰 범주이며, 프로퍼티는 메소드에 속한다고 볼 수 있다. 프로퍼티는 메소드의 일종으로 get과 set으로 구분된다. 프로퍼티는 메소드로 만들 수 있다.(형식적으로) 해당 C#에서는 아예 메소드화 시켰으며, 필드처럼 사용되게 편의성을 높인것이 특징이다.

##### (7) set-accessor vs. get-accessor

- set-accessor : 셋 접근자라고 불리며, 값을 지정할 때 사용, value라는 지정어 사용, 매개변수 역할을 하며 배경 연산의 우측식이 연산결과를 전달받는다.

get-accessor : 갓 접근자, 값을 받아올 때(참조할 때) 사용한다.

##### (8) operator overloading vs. method overloading

- operator overloading(연산자 중복) : (기존 자료형과 연산자들의 의미를 재정의하는 것)

시스템에서 제공한 연산자를 특정 클래스만을 위한 연산의 의미를 갖도록 재정의하는 것, 자료 추상화를 이루며, 유지보수가 쉬워진다.

method overloading(메소드 중복) : 메소드의 시그니처를 다르게 하여 이름을 같으나 다른 메소드처럼 사용하는 것(이름 동일, 개수와 매개변수 형 다름) 즉, 메소드를 매개변수에 따라 선택하게 하기위한 방법 중 하나이다.

##### (12) class vs. struct

- 클래스와 구조체 거의 동일하지만, 가장 큰 차이점은 클래스는 참조형, 구조체는 값형이다. 클래스는 힙에, 구조체는 스택에 저장되며, 구조체는 사웃 ㄱ이 불가능하며, 구조체는 생성자, 소멸자 개념이 존재하지않는다.그러므로 당연하게도 구조체에는 멤버의 초기값 개념을 부여할 수 없다.

#### 4-5. 다음을 간략히 설명하시오.

##### (1) C# 언어에서 제공한 접근 수정자에 따른 참조 범위를 각각 설명하시오.

-public : 모든 클래스, 같은 클래스, 파생 클래스, 동일 네임스페이스, 다른 네임스페이스

private : 같은 클래스만

protected : 같은 클래스와 파생 클래스에서만

internal : 같은 클래스와 동일 네임스페이스서만

protected internal : 같은 클래스와 파생클래스 그리고 동일 네임스페이스

##### (2) 메소드 중복에서, 매개변수의 개수가 같고 매개변수의 형이 다른 경우에 메소드를 구별하는 방법을 설명하시오.

- 매개변수의 개수가 같고 매개변수의 형이 다른 경우 -> 컴파일러가 해당 메소드를 선택하고 해당하는 형이 존재하지 않으면 기본적인 형 변환에 의해 실 매개변수의 자료형을 형식 매개변수의 자료형으로 바꿀 수 있는 메소드를 선택한다.(p174)

##### (3) C# 언어의 프로퍼티에서 사용 가능한 접근 수정자를 설명하시오.

- 프로퍼티의 수정자 = 메소드의 수정자이다. 종류는 new, static, virtual, sealed, override, abstract, extern(p.181)이 있다.

##### (4) C# 언어에서 연산자 중복하는 방법을 설명하시오.

- public static [extern] 리턴타입 operator (연산자형태)(매개변수1, 매개변수2) { ... 몸통... ;}

여기서 주의할 점은 반드시 public static 접근자로 지정해야하며, 연산자 형태는 +, -, [, ] 연산자도 가능하다.

##### (5) 사용자 정의 형 변환(user-defined type conversion)이란 무엇인가?

- 사용자 정의 형 변환은 클래스나 객체나 구조체를 다른 클래스 객체나 구조체 또는 C# 기본 자료형으로 변환하는 기능으로 해당 변환을 직접 사용자가 정의하는 것을 의미한다. (객체, 클래스 <----> 다른 클래스 객체, 클래스, 기본 자료형) 단항 연산자이므로 매개변수는 항상 1개, 매개변수 형은 변환을 하고 싶은 형태를 적어야한다. 사용자 정의 형 변환은 반드시 explicit(명시적), implicit(묵시적) 중 하나로 정의되어야한다.

4-6. 다음 C# 프로그램에는 에러가 있다. 그 이유를 밝히시오.

4.6 다음 C# 프로그램에는 에러가 있다. 그 이유를 밝히시오.

```
using System;
class Dummy {
    const int MIN = MAX + 100;
    const int MAX = MIN + 100;
}
```

MAX가 정의되기 전에 MIN 값을 정의  
⇒ 순환 정의 오류

CHAPTER 4 클래스 213

```
public static readonly int NORMAL;
public int instanceVariable;
static int staticVariable;
}
class ExerciseCh4_6 {
    public static void Main() {
        Dummy d = new Dummy();
        Dummy.NORMAL = 5;
        d.staticVariable = 10;
        Dummy.instanceVariable = 10;
        Dummy.staticVariable = 10;
    }
}
```

NORMAL은 static readonly로 선언되었으므로 수정 불가능! (5)시, 오류!  
Static Variable은 객체별로 참조된다! 수정: Dummy.staticVariable = 10;  
객체명으로 참조받은 것은 객체 참조! 수정: d.instanceVariable = 10;  
객체로 참조해야 함!

4-7. 다음 프로그램의 실행 결과를 쓰시오.

4.7 다음 프로그램의 실행 결과를 쓰시오.

(1)

```
using System;
class Exercise4_7_1 {
    static void Inc(ref int x, int y) {
        ++x; ++y;
        Console.WriteLine(" Inc: x = {0}, y = {1}", x, y);
    }
    public static void Main() {
        int x = 1, y = 1;
        Console.WriteLine("Before: x = {0}, y = {1}", x, y);
        Inc(ref x, y);
        Console.WriteLine(" After: x = {0}, y = {1}", x, y);
    }
}
```

<결과>  
Before: x=1, y=1  
Inc : x=2, y=2  
After: x=2, y=1  
↳ why? x만 ref 매개변수 수정  
y는 값 전달!

#### 4-8. 다음과 같은 프로퍼티와 동등한 기능을 수행하는 메소드로 바꾸시오.

4.8 다음과 같은 프로퍼티와 동등한 기능을 수행하는 메소드로 바꾸시오.

<pre>public int Numerator {     get { return numerator; }     set { numerator = value; } }</pre>	$\Rightarrow$	<pre>private int numerator; public int getNumerator() { return numerator; } public void setNumerator(int numerator) {     this.numerator = numerator; }</pre>
--	---------------	---

### <4장 : 실습문제>

(1) 4-11 소스코드

( 분수 생성자 부분 )

```
test
7 namespace test
8 {
9     //분수 클래스
10    참조 29개
11    class Fraction
12    {
13        public int numerator;//분자
14        public int denominator;//분모
15
16        //(1) 한 개의 정수를 받아 초기화하는 생성자
17        참조 4개
18        public Fraction(int numerator)
19        {
20            this.numerator = numerator;
21        }
22
23        //(2) 두 개의 정수를 받아 초기화 하는 생성자
24        참조 3개
25        public Fraction(int numerator, int denominator)
26        {
27            this.numerator = numerator;
28            this.denominator = denominator;
29        }
30    }
31 }
```

( 분수 표현하기 위한 오버라이딩 부분)

```
26 // (3) 하나의 분수를 분자/분모 형태로 변환하는 ToString() 메소드
27 참조 0개
28 override public string ToString()//public 과 override 위치는 바뀌어도 상관없다
29 {
30     return (numerator + "/" + denominator);
31     //return base.ToString(); 원래 해당 문구로 인해 오리지널 출력값이다. 여기서 우리가 override시키는 것이다!
32 }
33
34 //(4) 최대공약수를 구하는 메소드 & 기약 분수로 만드는 메소드
35 //최대공약수-> 두 수 이상의 공통된 약수를 의미
36 // 최대공약수 = 두 수 곱한 것 / 최소 공배수 이므로 최소 공배수 구하는 메소드 작성
37 참조 1개
```



( 기약분수를 만들기 위한 최대공약수 함수 (클래스 내에 선언) )

```
36 public int gcd(int num1, int num2)
37 {
38
39     int max = num1; // 큰수로 가정 max는 최대공약수를 담을 필드
40     int min = num2; // 작은 수로 가정 min은 최소 공배수를 담을 필드
41     int temp;      // 값을 계속 옮긴 임시 변수
42
43     while (true)
44     {
45         //(1)
46         temp = max % min; //큰 수에서 작은 수를 0이 될때까지 나누면 최소공배수
47         //이 때 0이라는 뜻은 딱 나누어 떨어졌다는 뜻!
48
49         //(2)
50         max = min; // 반복하기 위한 방법1
51
52         //(pause-조건체크)
53         if (temp == 0) //0이 되었을 때의 탈출 조건
54         {
55             break;
56         }
57         //(3) -> (1)로!
58         min = temp; // 해당 부분 배치 이유 : 0이 딱 되었을 때 min에 들어가는 값은 최소공배수인데 min=temp 시, min은 반드시 0이 됨
59     }
60
61     // 탈출했을 때의 min은 최대공약수
62     //max = (num1 * num2) / min;
63
64     return min;
```

( 기약분수를 만드는 메소드 )

```
//기약분수 만드는 메소드-> 분자와 분모의 최대공약수로 나눈다면 구할 수 있다.
참조 4개
public void reducedFraction(Fraction f)
{
    int gcd_value = gcd(f.numerator, f.denominator); //최대 공약수 구하기
    f.numerator = f.numerator / gcd_value;           //그걸로 약분 하기~!
    f.denominator = f.denominator / gcd_value;
}
```

( 분수 덧셈 메소드 )

```
77
78 // (5) 분수에 대한 사칙 연산을 수행하는 메소드
79
80 // (5-1) AddFraction
참조 1개
81 public Fraction AddFraction(Fraction f1, Fraction f2)
82 {
83     Fraction result = new Fraction(1); //임의의 결과를 받을 분수 받기
84
85     int result_numerator;
86     int result_denominator;
87
88     result_numerator = f1.numerator * f2.denominator + f1.denominator * f2.numerator;
89     result_denominator = f1.denominator * f2.denominator;
90
91     result.numerator = result_numerator;
92     result.denominator = result_denominator;
93
94     result.reducedFraction(result); //기약분수로 나누는 알고리즘으로 식을 정리
95     return result; //결과 분수 반환
96 }
97
```

(분수 뺄셈 메소드)

```
98
99 // (5-2) SubFraction
100 참조 1개
101 public Fraction SubFraction(Fraction f1, Fraction f2)
102 {
103     Fraction result = new Fraction(1); // 임의의 결과를 받을 분수 받기
104
105     int result_numerator;
106     int result_denominator;
107
108     result_numerator = f1.numerator * f2.denominator - f1.denominator * f2.numerator;
109     result_denominator = f1.denominator * f2.denominator;
110
111     result.numerator = result_numerator;
112     result.denominator = result_denominator;
113
114     result.reducedFraction(result); // 기약분수로 나누는 알고리즘으로 식을 정리
115     return result;
116 }
```

(분수 곱셈 메소드)

```
116
117 // (5-3) MulFraction
118 참조 1개
119 public Fraction MulFraction(Fraction f1, Fraction f2)
120 {
121     Fraction result = new Fraction(1, 1); // 임의의 결과를 받을 분수 받기
122
123     int result_numerator;
124     int result_denominator;
125
126     result_numerator = f1.numerator * f2.numerator;
127     result_denominator = f1.denominator * f2.denominator;
128     result.numerator = result_numerator;
129     result.denominator = result_denominator;
130
131     result.reducedFraction(result); // 기약분수로 나누는 알고리즘으로 식을 정리
132     return result;
133 }
134
```

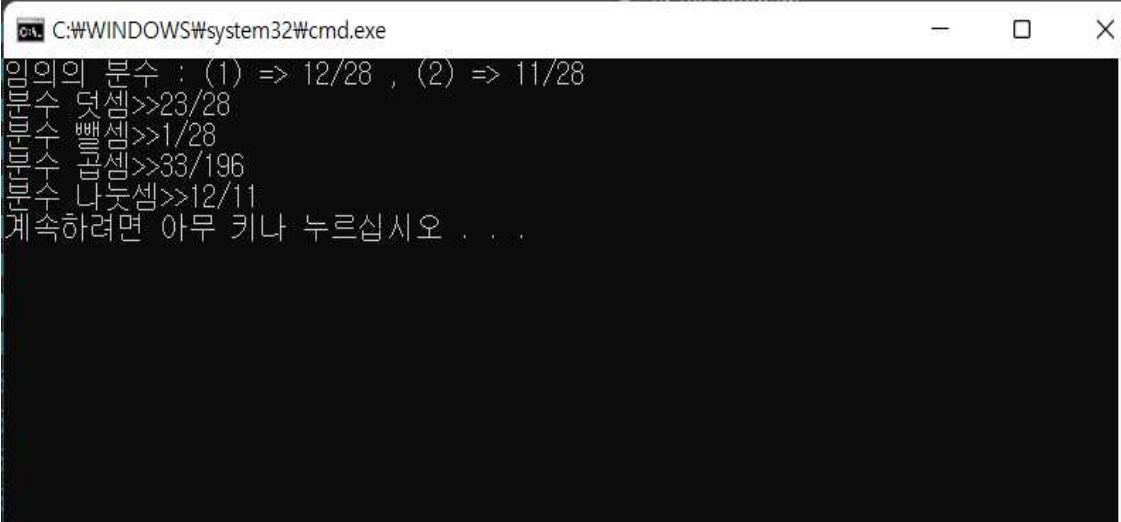
(분수 나눗셈 메소드)

```
134
135 // (5-4) DivFraction
136 참조 1개
137 public Fraction DivFraction(Fraction f1, Fraction f2)
138 {
139     Fraction result = new Fraction(1); // 임의의 결과를 받을 분수 받기
140
141     int result_numerator;
142     int result_denominator;
143
144     result_numerator = f1.numerator * f2.denominator;
145     result_denominator = f1.denominator * f2.numerator;
146
147     result.numerator = result_numerator;
148     result.denominator = result_denominator;
149
150     result.reducedFraction(result); // 기약분수로 나누는 알고리즘으로 식을 정리
151     return result;
152 }
153
154
```

( 메인 함수(동작결정) )

```
57 static void Main(string[] args)
58 {
59     Fraction result = new Fraction(1); //결과값을 받는 분수 생성 (임의의 분수)
60     Fraction num = new Fraction(12, 28); // 임의의 분수1
61     Fraction num2 = new Fraction(11, 28); // 임의의 분수2
62
63     Console.WriteLine("임의의 분수 : (1) => {0} , (2) => {1}", num, num2);
64
65     Console.Write("분수 덧셈>>");
66     result = result.AddFraction(num, num2);
67     Console.WriteLine(result);
68
69     Console.Write("분수 뺄셈>>");
70     result = result.SubFraction(num, num2);
71     Console.WriteLine(result);
72
73     Console.Write("분수 곱셈>>");
74     result = result.MulFraction(num, num2);
75     Console.WriteLine(result);
76
77     Console.Write("분수 나눗셈>>");
78     result = result.DivFraction(num, num2);
79     Console.WriteLine(result);
80
81
82
83
```

#### 4-11 결과화면



```
C:\WINDOWS\system32\cmd.exe
임의의 분수 : (1) => 12/28 , (2) => 11/28
분수 덧셈>>23/28
분수 뺄셈>>1/28
분수 곱셈>>33/196
분수 나눗셈>>12/11
계속하려면 아무 키나 누르십시오 . . .
```

## (2) 4-13 소스코드

### (스택 클래스 구현 : Push)

```
참조 4개
class Stack //스택의 기본구조를 따라감
{
    private int[] stack; //스택을 담을 배열
    int sp = -1; //스택의 제일 위를 표시할 수 (top이라고도 불림)
    참조 0개
    public Stack(int size) //생성 자 생성 시, 매개변수 있게 생성 때, 해당 매개변수만큼 배열을 동적할당
    {
        stack = new int[size];
    }

    //(1) 디폴트 생성자 생성(기본 크기는 100)
    참조 1개
    public Stack() //생성자 생성 시, 매개변수 없이 생성할 때, 100개를 기본으로 동적할당
    {
        stack = new int[100]; //int[] stack = new int[100] 과 동일 표현
    }

    참조 1개
    public void Push(int data) //Push 메소드
    {
        if (sp == stack.Length) //만약 sp가 스택 배열의 제일 위 라고 한다면
        { //스택이 꽉찼다면
            Console.WriteLine("스택이 꽉 찼습니다."); //오류문구 출력
        }
        else //일반 상황...
        {
            sp++; //sp를 1 증가(최상위층 번호 올리기)
            stack[sp] = data; //해당 부분에 데이터 삽입(push)
        }
    }
}
```

### (스택 클래스 구현 : Pop)

```
참조 1개
public int Pop() //Pop 메소드
{
    int sp_data; //pop 되는 임의의 데이터를 받을 int 임시 변수
    if (sp == -1)
    { //스택이 비었다면
        Console.WriteLine("스택이 비었습니다.");
        return 0;
    }
    else //스택이 비지않는 일반적인 상황
    {
        sp_data = stack[sp]; //pop된 데이터를 임시 필드 sp_data에 삽입
        sp--; //sp를 감소시켜 아래로 내려감을 표시
        return sp_data; //pop되는 데이터는 리턴으로
    }
}

참조 1개
public int getsp() { return sp; } //꼭대기 참조, main문의 루프를 위해(sp의 보안상때문에 접근용으로!)
//only read 용도

참조 0개
```

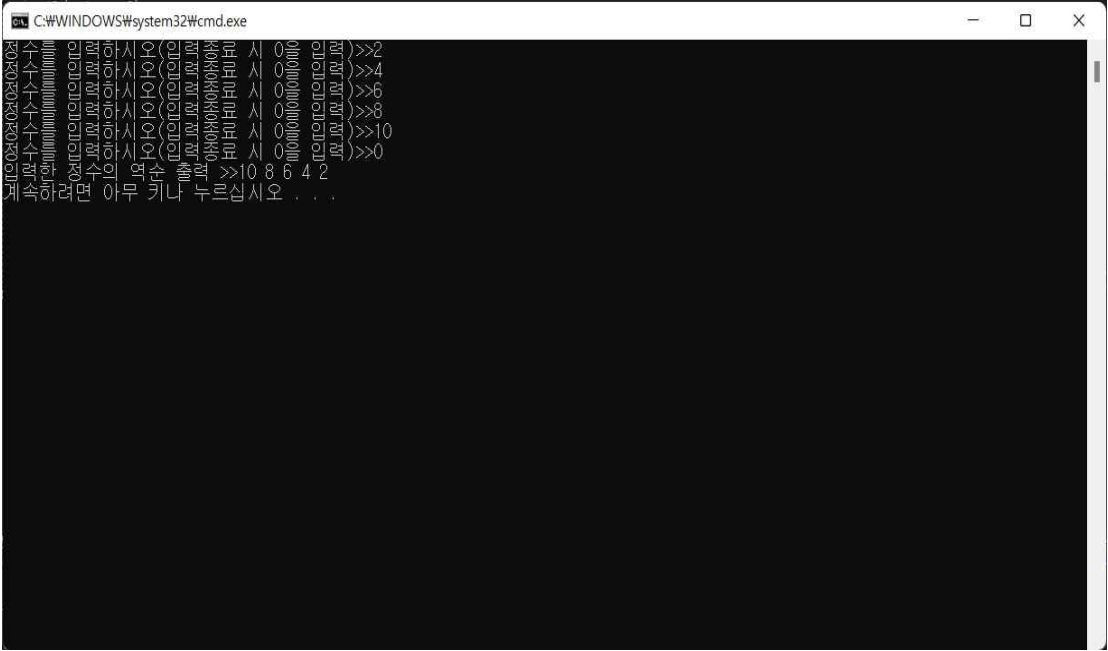
### (메인 함수: 테스트)

```
참조 0개
static void Main(string[] args)
{
    Stack reverse_integer = new Stack(); //배열이 100인 스택 생성
    int number; //입력 받을 숫자
    while (true)
    {
        Console.Write("정수를 입력하십시오(입력종료 시 0을 입력)>>");
        number = int.Parse(Console.ReadLine()); //int형으로 받기 위해 int.Parse()로 ReadLine값싸주기
        if (number == 0)
        {
            break;
        }
        else
        {
            reverse_integer.Push(number); //스택에 값 그대로 푸시(00이 나올때까지)
        }
    }

    Console.WriteLine("입력한 정수의 역순 출력 >>");
    //제일 위를 가르키는 값을 받아서 pop()할때마다 값을 출력하는 방식 채택
    //즉, 푸시된 스택을 그대로 빼어서 출력하면 그것이 역순 출력이 되어진다.(먼저 들어간건 제일 마지막에 나오는 스택의 특성)
    for (int i = reverse_integer.getsp(); i >= 0; i--) //초기화 값 (값을 getsp()메소드를 통해서 제일 위에있는 값을 가져옴)
    {
        Console.Write(reverse_integer.Pop() + " "); //하나씩 출력
    }
    Console.WriteLine(""); //개행
}
```



4-13 결과화면



### (3) 4-16 소스코드

(기본 필드 및 생성자 선언)

```
7 namespace test
8 {
9     참조 26개
10    class Complex
11    {
12        public double real; //실수부
13        public double image; //허수부
14
15        //예제 4-12에 있는 부분 참조
16        참조 4개
17        public Complex(double temp)
18        {
19            real = temp;
20            image = 0;
21        }
22
23        참조 2개
24        public Complex(double real, double image)
25        {
26            this.real = real;
27            this.image = image;
28        }
29    }
30 }
```

(연산자 중복 +, -)

```
26 // (1) 사칙연산 중복 정의 -> 핵심은
27 // public static으로 선언, 이항 연산자는 매개변수는 2개이며, 한개는 반드시 자신이 클래스 형태여야하고
28 // 반환형은 모든자료형이 가능하다.
29 참조 2개
30 public static Complex operator +(Complex x, Complex y)
31 {
32     //임의의 result 객체를 이용해 값을 넘겨받아 리턴한다.
33     Complex result = new Complex(0);
34     result.real = x.real + y.real;
35     result.image = x.image + y.image;
36
37     return result;
38 }
39
40 참조 2개
41 public static Complex operator -(Complex x, Complex y)
42 {
43     Complex result = new Complex(0);
44     result.real = x.real - y.real;
45     result.image = x.image - y.image;
46
47     return result;
48 }
```

(연산자 중복, \*, /)

```
47 참조 2개
48 public static Complex operator *(Complex x, Complex y)
49 {
50     Complex result = new Complex(0);
51     result.real = x.real * y.real - x.image * y.image;
52     result.image = x.real * y.image + x.image * y.real;
53
54     return result;
55 }
56
57 public static Complex operator /(Complex x, Complex y)
58 {
59     Complex result = new Complex(0);
60     result.real = (x.real * y.real + x.image * y.image) / ((y.real * y.real) + (y.image * y.image));
61     result.image = ((x.image * y.real) - (x.real * y.image)) / ((y.real * y.real) + (y.image * y.image));
62
63     return result;
64 }
```

(복소수 표현을 위한 ToString() 오버라이드)

```
64
65 // (2) 복소수를 위한 ToString 메소드
참조 0개
66 override public string ToString() // public 과 override 위치는 바뀌어도 상관없다
67 {
68     // Console.WriteLine(Complex 객체) 하면 아래의 형태로 출력된다.
69     return "(" + real + "," + image + ")"; // (실수부, 허수부) 형태
70 }
71
72 }
```

(Main 함수 부분)

```
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        // 임의의 복소수 2개 생성 (생성자로 초기화)
        Complex x = new Complex(1, 3);
        Complex y = new Complex(2, 4);
        Console.WriteLine("==테스트클래스 : 복소수 ==");
        Console.WriteLine("테스트할 복소수 >> {0} 과 {1} ", x, y); // ToString으로 이렇게해도 (실수부, 허수부)로 출력
        Console.WriteLine("덧셈 >> ({0})+({1})i", (x + y).real, (x + y).image); // (x+y) 시 결과값은 result -> result.real, result.image과 동일 값
        Console.WriteLine("뺄셈 >> ({0})+({1})i", (x - y).real, (x - y).image);
        Console.WriteLine("곱셈 >> ({0})+({1})i", (x * y).real, (x * y).image);
        Console.WriteLine("나눗셈 >> ({0})+({1})i", (x / y).real, (x / y).image);
    }
}
```

#### 4-16 결과화면

```
C:\WINDOWS\system32\cmd.exe
==테스트클래스 : 복소수 ==
테스트할 복소수 >> (1,3) 과 (2,4)
덧셈 >> (3)+(7)i
뺄셈 >> (-1)+(-1)i
곱셈 >> (-10)+(10)i
나눗셈 >> (0.7)+(0.1)i
계속하려면 아무 키나 누르십시오 . . .
```

## <5장 : 이론문제>

5.1 다음 괄호 안에 적당한 용어를 쓰시오.

### | 연습문제 |

5.1 다음 괄호 안에 적당한 용어를 쓰시오.

- (1) 클래스를 확장하여 새로운 클래스를 만들 수 있는 기능은 프로그램의 (재사용)을 높일 수 있다.   
⇒ 상속, 다중 상속
- (2) 이미 존재하는 클래스에 정보를 추가하여 새로운 클래스를 만들 수 있다. 이때 기존의 클래스를 (베이스클래스)라 부르고 새로 정의한 클래스를 (파생클래스)라 부른다.
- (3) 베이스 클래스로부터 필드를 상속받을 때, 파생 클래스에 같은 이름의 필드가 있으면 베이스 클래스의 필드는 숨겨진다. 이 경우에 (base) 라는 지정어와 함께 사용하면 베이스 클래스에 있는 필드를 참조할 수 있다.   
⇒ base 클래스와 관련!
- (4) 베이스 클래스로부터 메소드를 상속받을 때, 파생 클래스 내에 같은 이름의 메소드가 있는 경우에 시그네처가 다르면 (오버로딩)이 되고 시그네처가 같으면 (오버라이딩)이 된다.   
⇒ 이름, 개수까지 같아도,   
파생 클래스에 재정의하여 사용할 것을 암시
- (5) 지정어 virtual로 선언된 (가상) 메소드를 가상 메소드라 부른다.   
⇒ 가상 클래스에 재정의하여 사용할 것을 암시
- (6) 메소드 몸체(body)를 갖지 않는 메소드는 (추상메소드)와 (외부메소드)이다.   
⇒ 선언 메소드(최초 선언만 있!)   
⇒ 선언 메소드(최초 선언만 있!)
- (7) 실질적인 구현을 갖지 않고 메소드 선언만 있는 메소드를 (추상메소드)라 말한다.
- (8) 파생 클래스형의 객체가 베이스 클래스형으로 바뀌는 형 변환을 (업캐스팅)라 말한다.   
(다운캐스팅)
- (9) 메소드의 의미가 적용하는 객체에 따라 달라지는 것을 (다형성)이라 부른다.
- (10) 프로그래머가 자신이 정의한 클래스나 메소드가 파생 클래스에서 재정의되지 않기를 원한다면 (sealed)으로 선언하는 것이 바람직하다.   
(봉인 클래스 메소드)
- (11) C#에서, (인터페이스)는 메소드, 프로퍼티, 인덱서, 이벤트의 선언부만 갖고 있는 순수한 설계의 표현이다.   
⇒ C++에서 순수 가상 함수
- (12) Main() 메소드가 없는 어셈블리는 실행할 수 없고 (라이브러리) 형태가 된다.   
→ P250
- (13) 컴파일 과정에서 DLL을 명시하는 옵션은 (target:library)이다.   
→ target:library   
→ DLL로 만들어줘야 한다.   
타겟 라이브러리

5.2 다음 객관식 문제를 푸시오



5.2 다음 객관식 문제를 푸시오.

(1) 자료 추상화

(1) 자료 추상화  $\rightarrow$  자료의 기본  
(2) 다형성  $\rightarrow$  클래스

(b) 정보 은폐화  $\Rightarrow$  *인생*

② 상속성

④ 유지 보수성

⑤ 재사용성

③ 다행성 →  $\frac{1}{2}$ 의 수제에 해당함

⑤ 자료 추상화

② 상속성  $\rightarrow$  상속 관계  $\rightarrow$  상속 그 자체! 원본 (기)이다!

④ 정보 은닉화, 정보 보호

↳ 가계지출의 메커니즘 (목표)  
= 지출증

(3) 파생 클래스의 생성자가 실행되는 순서는?

필드 초기화 → 베이스 클래스 생성자 → 파생 클래스 생성자

② 파생 클래스 생성자 → 필드 초기화 → 베이스 클래스 생성자

③ 베이스 클래스 생성자 → 필드 초기화 → 파생 클래스 생성자

④ 베이스 클래스 생성자 → 파생 클래스 생성자 → 필드 초기화

⑤ 필드 초기화 → 파생 클래스 생성자 → 베이스 클래스 생성자

(\*) 수정자로 올 수 있는 지정어의 개수가 5개(접근 수정자와 new)가 아닌 것은?

### ① 구조체(struct) 수정자

② 인터페이스(interface) 수정자

③ 델리게이트(delegate) 수정자

④ 생성자(constructor) 수정자

⑤ 열거형(enum) 수정자 [ public

public  
 protected  
 internal  
 private

→ protected internal private new  
의미는 필드 생성 안함  
→ 객체 생성 시 new!

(5) 프로그래머가 자신이 정의한 클래스나 메소드가 파생 클래스에서 (재정의)되지 못하게

할 때 사용하는 지정어는?

① static → 정적 클래스, 메서드  
재정의는 X

③ private → 친구 걱정

⑤ readonly

② public - 78!

✓ sealed — 봉인클래즈, 메슨드 (자명일별가)

© 2000 Blackwell Science Ltd

(6) 메소드 수정자에 관한 설명 중 (맞는 것은)?

④ static, virtual, 그리고 override는 함께 사용할 수 있다.

② new와 override는 함께 사용할 수 있다. → new, static, final 한 가지

④ 예스드가 abstract로 선언되면, static, virtual, sealed 또는 e

한편 사용한 수 있다

✓ 메소드가 private로 선언되면, virtual, override, 또는 abstract

사용할 수 있다.

사용할 수 있다.

sealed & override는 함께 사용할 수 없다. 2.47

[illegible]

자원이 풍부한 환경에서

new 키워드는 기본값으로  
override 키워드로 다룰 수 있음!  
↑  
virtual을 쓴다!  
↳ 실용적이고 동시에 사용이 쉽!

$\triangle 2, 4, 3$

(7) 메소드 수정자에 관한 설명 중 옳지 않은 것은? 3

- 1. static으로 선언하면, 반드시 클래스 이름을 통해서만 참조할 수 있다. (정확한 수정)
- 2. extern으로 선언하면, C# 이외의 언어로 구현해야 한다. → 메소드가 C#도 외부에서 구현!
- 3. virtual로 선언하면, 반드시 new 키워드를 사용해야 한다. → override한 선택지 문제!
- 4. abstract로 선언하면, 메소드 몸체를 정의할 수 없다. → 추상클래스의 특징
- 5. sealed로 선언하면, 파생 클래스에서 재정의할 수 없다. → 본인 클래스에서!

(8) 메소드 수정자의 의미에 관한 설명 중 맞지 않은 것은? 4

- 1. 메소드가 virtual로 선언되면, 파생 클래스에서 반드시 new 수정자를 사용하여 재정의해야만 한다. → override 사용!
- 2. 메소드가 abstract로 선언되면, 파생 클래스에서 반드시 new 수정자를 사용하여 재정의해야만 한다. → override 사용!
- 3. 메소드가 virtual로 선언되면, 파생 클래스에서 반드시 override 수정자를 사용하여 재정의해야만 한다. → new 사용!
- 4. 메소드가 abstract로 선언되면, 파생 클래스에서 반드시 override 수정자를 사용하여 재정의해야만 한다.
- 5. 메소드가 sealed로 선언되면, 파생 클래스에서 반드시 override 수정자를 사용하여 재정의해야만 한다. → 재정의 불가!

BaseClass S = new DerivedClass();  
파생클래스에서  
new 수정자 사용시 ⇒ Base클래스에  
override 수정자 || ⇒ Derived클래스에  
미 기입 ⇒ Base클래스

(9) 재정의할 수 있는 메소드는? 2

- 1. 정적 메소드 접근, 구현 가능 / 재정의 불가!
- 2. 가상 메소드 → virtual → new 재정의 가능!
- 3. 봉인 메소드 sealed의 특징
- 4. 외부 메소드 (external method) → 재정의 불가!

(10) C#에서 인터페이스 몸체가 될 수 없는 것은? 5

- 1. 메소드 선언
- 2. 프로퍼티 선언
- 3. 인덱서 선언
- 4. 이벤트 선언
- 5. 델리게이트 선언 ⇒ C#의 함수형 언어 대응! 메소드를 가리키는 참조형. (메소드 변수 등..)

(11) C#에서 인터페이스 수정자(interface modifier)가 될 수 없는 것은? 5

- 1. public → default 수정
- 2. protected
- 3. internal
- 4. private
- 5. static → 클래스의 종적 속성 → interface는 변경 불가!

④ interface는 생성자 X

(12) C#에서 부분 선언이 가능하지 않은 프로그래밍 단위는 무엇인가? 4

- ① 클래스 (o)
- ② 구조체 (o)
- ③ 인터페이스 (o)
- ④ 모듈  $\Rightarrow$  .cs 단위

(13) C#에서 부분 선언으로 얻을 수 있는 장점은 무엇인가? 2

- ① 재사용성
- ② 융통성
- ③ 추상화
- ④ 효율성
- ⑤ 다형성

C#의 부분선언  $\Rightarrow$  partial 키워드  
= 하나의 형식은 여러 곳에 나누어 정의  
= 각각 다른 클래스에서 자동 생성 클래스  $\rightarrow$  직접 추가 가능  
필요 code

(14) 인터페이스 멤버에 관한 설명 중 틀린 것은?

- ① 메소드, 프로퍼티, 인덱서, 이벤트가 될 수 있다. (o)
- ② 인터페이스 메소드는 모두 추상 메소드가 된다.  $\rightarrow$  명시적 abstract 속성!
- ③ 선언으로만 구성되고 구현을 가질 수 없다. (o)  $\rightarrow$  옳음 X
- ④ 접근 수정자는 명시적으로 internal이 된다.  $\rightarrow$  public
- ⑤ 인터페이스는 생성자를 가질 수 없다.  $\rightarrow$  생성자가 abstract이므로!

(15) C# 프로그래밍에서 다중 상속이 가능한 프로그래밍 단위는? 3

- ① 클래스  $\rightarrow$  단일 상속만 가능!
- ② 구조체  $\rightarrow$  상속?  $\rightarrow$  불가
- ③ 인터페이스  $\rightarrow$  여러 개에 인터페이스 지정 가능
- ④ 네임스페이스 X
- ⑤ 어셈블리 X  $\rightarrow$  다중 상속 (옳음)

(16) 인터페이스와 추상 클래스의 설명 중 맞는 것은? 1

- ① 인터페이스는 다중 상속을 지원하나 추상 클래스는 단일 상속만 가능하다. (o)
- ② 인터페이스는 객체를 가질 수 없지만 추상 클래스는 객체를 가질 수 있다. 객체 X (x)
- ③ 인터페이스는 메소드 선언만 있을 수 있고 추상 클래스는 오직 추상 메소드만 가질 수 있다.  $\rightarrow$  프로퍼티 인덱서 이벤트 등에서 객체  $\rightarrow$  그냥 메소드만 but 객체 생성 X (x)
- ④ 인터페이스는 메소드 정의를 가질 수 있지만 추상 클래스는 오직 메소드 선언만 있을 수 있다. 선언만!
- ⑤ 인터페이스의 최상위 인터페이스는 System.Interface이고 추상 클래스의 최상위 클래스는 System.Object이다.

System.Object  
 $\rightarrow$  C#의 모든 타입은 해당 클래스로 파생



각종 다른 개념에 따라 메소드 등의 의미가 달라지는 것  
핵심: virtual & override 개념.

(17) C# 프로그래밍에서 다형성을 이루려면?

1. (1) 베이스 클래스에서는 virtual로 선언하고 파생 클래스에서는 override로 재정의해야 한다.
- (2) 베이스 클래스에서는 virtual로 선언하고 파생 클래스에서는 new로 재정의해야 한다.   
 (Base 클래스 메소드가 재정의)
- (3) 베이스 클래스에서는 static으로 선언하고 파생 클래스에서는 override로 재정의해야 한다.   
 (재정의 X)
- (4) 베이스 클래스에서는 static으로 선언하고 파생 클래스에서는 new로 재정의해야 한다.   
 (static 다형성 X (재정의 X))
- (5) 베이스 클래스에서는 abstract로 선언하고 파생 클래스에서는 메소드 수정자 없이 재정의해야 한다.   
 (virtual의 의미가 없음) override 수정자 필요!

(18) 메소드의 몸체를 가질 수 없는 메소드는 어느 것인가?

- ① 정적 메소드와 봉인 메소드 X
  - ② 가상 메소드와 봉인 메소드 X
  - ③ 가상 메소드와 외부 메소드 X
  - ④ 추상 메소드와 가상 메소드 X
  - ⑤ 추상 메소드와 외부 메소드
5. 가상(virtual) 메소드  
→ 몸체 있음, new할 경우 Base 클래스의 virtual 메소드 이용.  
외부 메소드의 메소드 수정자가 없거나  
봉인  
→ 몸체 있음, 재정의 X

(19) 클래스형 변환에 대한 내용 중 바른 것은?

- ① 베이스 클래스형은 파생 클래스형으로 묵시적으로 변환된다. (캐스트-다운)
  - ② 베이스 클래스형은 파생 클래스형으로 묵시적으로 변환해야 한다. (캐스트-업)   
 (C#에서는 예외!)
  - ③ 파생 클래스형은 베이스형으로 묵시적으로 변환된다. (캐스트-업)
  - ④ 파생 클래스형은 베이스형으로 묵시적으로 변환해야 한다.   
 (but 명시적으로 변환해야 한다. 예: (BaseType) obj)
  - ⑤ 파생 클래스형 객체 참조가 베이스형 객체를 가리킬 수 있다.   
 (예: (BaseType) obj)
- 기타 메모: 캐스트-업 & 캐스트-다운, 베이스 클래스 형, 파생 클래스 형, 이란, 캐스트-업! (2, 3, 4, 5)

(20) 클래스의 계층적 구조에서 다음 중 올바른 내용을 모두 찾으시오. (2, 4, 5)

- ① 상위 계층에 있는 클래스일수록 특정한 응용에 적합한 클래스이다.
- ② 상위 계층으로 갈수록 추상 클래스가 많다.   
 (더 추상적임! (20개의 생략에 적합해함))
- ③ 상위 계층에 있는 메소드가 하위 계층에 있는 메소드를 재정의한다. → 반대!
- ④ 하위 계층으로 갈수록 좀더 일반적인 클래스이다. → 우리가 원했던 것만 반대!
- ⑤ 하위 계층의 형을 상위 계층의 형으로 변환할 수 있다.   
 (캐스트-업!)

(21) 다음 중 네임스페이스에 대한 설명 중 옳은 것은?   
 다 맞음!

- ① 서로 관련된 클래스나 인터페이스, 구조체, 열거형, 델리게이트, 그리고 서브 네임스페이스를 하나의 단위로 묶는 방법이다.
- ② 네임스페이스를 포함하기 위해서는 직접 namespace를 사용한다.
- ③ 클래스 이름 사이의 충돌 문제를 해결할 수 있다. → C++의 대안적 (1등: 마이클, 2등: 마이클)
- ④ System 네임스페이스는 using 문을 사용하지 않는다. ?
- ⑤ using 문을 사용하지 않으면 전체 이름을 써야 한다. (p250)

5.3 다음 구문에 대한 문법적인 구조를 기술하시오.

(1) 파생 클래스의 정의 형태

Shape, Rectangle, rec ...   
 이렇게!



## <5장 : 실습문제>

(1) 5-8 소스코드

(도형을 구상하기 위한 추상클래스)

```
참조 2개
abstract class Fig//추상클래스 선언
{
    참조 4개
    abstract public void Area();//넓이 추상클래스
    참조 4개
    abstract public void Girth();//둘레 추상클래스
    참조 4개
    abstract public void Draw();//그리기 추상클래스
}
```

(추상클래스를 이용한 사각형 클래스 구현)

```
19      참조 4개
20      class Rect : Fig //추상클래스 상속(구현해야함)
21      {
22          int width;
23          int height;
24
25          참조 0개
26          public Rect()//기본생성자
27          {
28              width = 1;
29              height = 1;
30          }
31          참조 1개
32          public Rect(int width, int height)//매개변수가 있는 생성자
33          {
34              this.width = width;
35              this.height = height;
36          }
37          참조 2개
38          public override void Area()//추상클래스 구현
39          {
40              Console.WriteLine("넓이 : " + width * height);
41          }
42          참조 2개
43          public override void Girth()//추상클래스 구현
44          {
45              Console.WriteLine("둘레 : " + (width + height));
46          }
47          참조 2개
48          public override void Draw()//추상클래스 구현
49          {
50              Console.WriteLine("====사각형====");
51          }
52      }
```

(추상클래스를 이용한 원 클래스 구현)

```
참조 4개
class Circle : Fig//추상클래스 상속(원을 구현해야함)
{
    double rad;

    참조 0개
    public Circle()//생성자
    {
        rad = 1.0;
    }

    참조 1개
    public Circle(double rad)//매개변수있는 생성자
    {
        this.rad = rad;
    }

    참조 2개
    public override void Area()//추상클래스 구현
    {
        Console.WriteLine("넓이 : " + (3.14 * rad * rad));
    }

    참조 2개
    public override void Girth()//추상클래스 구현
    {
        Console.WriteLine("둘레 : " + (2 * 3.14 * rad));
    }

    참조 2개
    public override void Draw()//추상클래스 구현
    {
        Console.WriteLine("====원====");
    }
}
```

(메인 함수)

```
참조 0개
static void Main(string[] args)
{
    Rect a = new Rect(10, 20);
    Circle b = new Circle(5);
    Console.WriteLine("추상클래스를 이용한 테스트프로그램 >>");

    //테스트 프로그램
    Console.WriteLine(" ");
    a.Area();
    a.Girth();
    a.Draw();

    Console.WriteLine(" ");
    b.Area();
    b.Girth();
    b.Draw();
}
```

## 5-8 결과화면

```
C:\WINDOWS\system32\cmd.exe
추상클래스를 이용한 테스트프로그램 >>

높이 : 200
바레 : 30
====사각형====

높이 : 78.5
바레 : 31.4
=====원=====
계속하려면 아무 키나 누르십시오 . . .
```

(2) 5-9 소스코드

(인터페이스 코드)

```
참조 2개
interface IOperation
{
    void Insert(string str);
    참조 4개
    string Delete();
    참조 4개
    bool search(string str);
    참조 5개
    string GetCurrentElt();
    참조 4개
    int NumOfElements();
}
참조 3개
```

(인터페이스 구현 : 스택)

```
22 public Stack()
23 {
24     stacking = new string[10]; //생성자 생성시 10개의 스택 생성
25 }
26 //우선 비었을 때, 꼭 찾을 때, 무시, 임의로 10개의 스택 생성
참조 2개
27 public string Delete()
28 {
29     return stacking[top--];
30 }
참조 2개
31 public string GetCurrentElt()
32 {
33     return stacking[top]; //현재 top에 있는 원소 반환
34 }
참조 4개
35 public void Insert(string str)
36 {
37     stacking[++top] = str;
38 }
참조 2개
39 public int NumOfElements()
40 {
41     return (top + 1); //top은 실질적으로 인덱스 값 , 그러므로 +1로 하여 실 갯수 맞춤
42 }
참조 2개
43 public bool search(string str)
44 {
45     for (int i = 0; i <= top; i++)
46     {
47         if (stacking[i] == str)
48             return true; //동일한게있으면 true.
49     }
50     return false;
51 }
52 }
```

## (인터페이스 구현 : 큐)

```
참조 3개
54 class Queue : IOperation
55 {
56     private string[] que_user;
57     int rear = -1;
58     int front = -1; //rear은 데이터가 들어가는 후면, front는 데이터가 나가는 면이 전면
59     참조 1개
60     public Queue()
61     { que_user = new string[10]; } //생성자 생성 시, 큐 '담을' 배열 생성
62     참조 2개
63     public string Delete()
64     {
65         //front = front + 1;
66         return que_user[++front]; //front에 있는 값 return 주기(1 증가 시킨 값)
67     }
68     참조 3개
69     public string GetCurrentElt()
70     { return que_user[front + 1]; } //값만 반환
71     참조 4개
72     public void Insert(string str)
73     { que_user[++rear] = str; } //rear를 한칸 전진 + 전진한 rear 인덱스에 데이터 삽입
74     참조 2개
75     public int NumOfElements()
76     { return ((rear - front + 10) % 10); } //값 마이너스를 막기 위한 모듈러 연산을 이용해 갯수를 세기
77     참조 2개
78     public bool search(string str)
79     {
80         for (; front <= rear + 1; front++)
81         {
82             if (que_user[front] == str)
83                 return true;
84         }
85         return false;
86     }
87 }
```

## (메인함수 : 스택 테스트)

```
참조 0개
static void Main(string[] args)
{
    Stack s = new Stack();
    //스택 테스트
    Console.WriteLine("=====스택테스트=====");
    //insert 테스트
    s.Insert("o");
    s.Insert("love");
    s.Insert("OMG");
    //GetCurrentElt 테스트
    Console.WriteLine("현재 최상층에 있는 원소확인 : " + s.GetCurrentElt());
    //Delete 테스트
    Console.WriteLine("방금 삭제된 원소 : " + s.Delete());
    //NumOfElements 테스트
    Console.WriteLine("스택에 존재하는 원소의 갯수(예상:2) : " + s.NumOfElements());
    //Search 테스트
    Console.WriteLine("현재 love라는 단어가 스택에 있는가? 있으면 true, 없으면 false : " + s.search("love"));

    Console.WriteLine("");
}
```

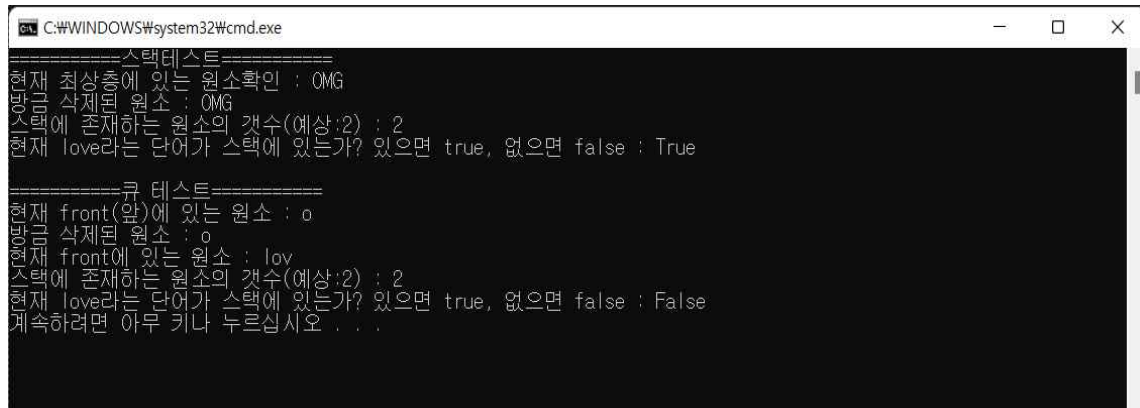
## (메인함수 : 큐 테스트)

```
//que 테스트

Console.WriteLine("=====큐 테스트=====");
Queue q = new Queue();
//insert 테스트
q.Insert("o");
q.Insert("love");
q.Insert("OMG");
//GetCurrentElt 테스트
Console.WriteLine("현재 front(앞)에 있는 원소 : " + q.GetCurrentElt());
//Delete 테스트
Console.WriteLine("방금 삭제된 원소 : " + q.Delete());
Console.WriteLine("현재 front에 있는 원소 : " + q.GetCurrentElt()); //delete 후 삭제된 원소 확인
//NumOfElements 테스트
Console.WriteLine("스택에 존재하는 원소의 갯수(예상:2) : " + q.NumOfElements());

//Search 테스트
Console.WriteLine("현재 love라는 단어가 스택에 있는가? 있으면 true, 없으면 false : " + q.search("love"));
```

## 5-9 결과화면



```
C:\WINDOWS\system32\cmd.exe

=====스택테스트=====
현재 최상층에 있는 원소확인 : OMG
방금 삭제된 원소 : OMG
스택에 존재하는 원소의 갯수(예상:2) : 2
현재 love라는 단어가 스택에 있는가? 있으면 true, 없으면 false : True

=====큐 테스트=====
현재 front(앞)에 있는 원소 : o
방금 삭제된 원소 : o
현재 front에 있는 원소 : lov
스택에 존재하는 원소의 갯수(예상:2) : 2
현재 love라는 단어가 스택에 있는가? 있으면 true, 없으면 false : False
계속하려면 아무 키나 누르십시오 . . .
```

### ■ 수업 성찰(배운점·느낀점)

해당 수업활동일지를 작성하며 C# 프로그래밍의 기본 기능과 더불어 C#에만 있는 독특한 기능들(프로퍼티, 형 변환 연산자 등)을 많이 알수 있는 기회가 되었다.