

## <4.7>

다음 프로그램의 실행 결과를 쓰시오.

(1)

```
using System;

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

(2)

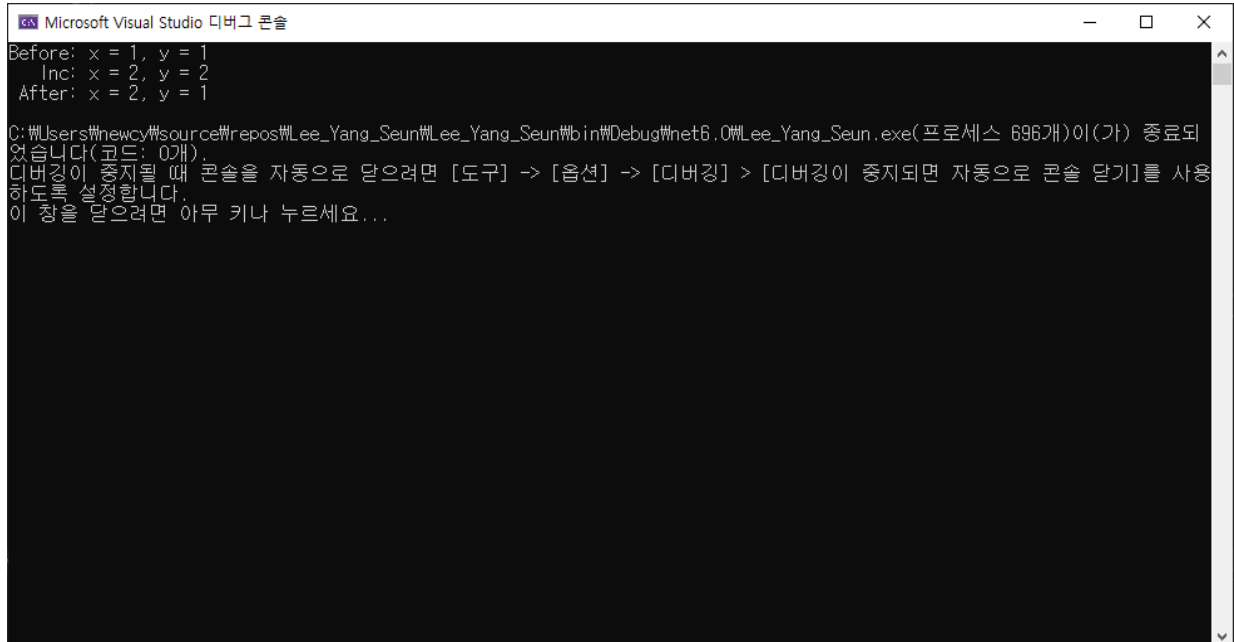
```
using System;
delegate void ExDelegator(string str);

참조 3개
class Ex {
    참조 1개
    public static void StaticMethod(string str)
    {
        Console.Write(str);
        Console.WriteLine("\nFramework");
    }
    참조 1개
    public void InstanceMethod(string str)
    {
        Console.Write(str);
        Console.WriteLine("\nProgramming Language");
    }
}

참조 0개
class ExerciseCh4_7_2
{
    참조 0개
    public static void Main()
    {
        Ex ex = new Ex();
        ExDelegator d1 = new ExDelegator(ex.InstanceMethod);
        ExDelegator d2 = new ExDelegator(Ex.StaticMethod);
        d1("C#");
        d2(".NET");
    }
}
```

## <결과>

(1)



Microsoft Visual Studio 디버그 콘솔

```
Before: x = 1, y = 1
Inc: x = 2, y = 2
After: x = 2, y = 1
```

C:\Users\newcy\source\repos\Lee\_Yang\_Seun\Lee\_Yang\_Seun\bin\Debug\net6.0\Lee\_Yang\_Seun.exe(프로세스 696개)이(가) 종료되었습니다(코드: 0개).

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...

(2)



Microsoft Visual Studio 디버그 콘솔

C# Programming Language  
.NET Framework

```
C:\Users\newcy\source\repos\Lee_Yang_Seun\Lee_Yang_Seun\bin\Debug\net6.0\Lee_Yang_Seun.exe(프로세스 11768개)이(가) 종료되었습니다(코드: 0개).
```

디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

이 창을 닫으려면 아무 키나 누르세요...

## <설명>

- 위의 코드는 값 호출과 참조 호출 시의 차이점을 보여주고 있다. Inc 함수는 x변수를 주소로

받아 ++x 연산 후 main함수에 전달하고, y변수는 사본의 형식으로 받아 ++y 연산 후에도 main함수에 값이 전달되지 않아 그대로 출력되는 결과를 확인할 수 있다.

- ExDelegator은 델리게이트로 정의되었고, Ex 클래스는 static void 함수의 StaticMethod 메소드와 void 함수의 InstanceMethod 메소드로 구성 되어있다. 두 메소드는 string 문자열을 매개변수로 입력 받아 출력하는 형태이다. Main 메소드에서는 Ex클래스를 정의하고, d1 델리게이트 객체와 d2 델리게이트 객체를 생성하였다. d1 객체는 Main 메소드에서 선언된 객체를 매개변수로 하여 InstanceMethod를 호출하였고, d2 객체는 클래스 명으로 StaticMethod 함수를 호출하였다.

## <4.11>

(1) 한 개의 정수를 받아 초기화하는 생성자를 작성하시오

```
public Fraction(int a)
{
    numerator = a;
    denominator = 1;
}
```

(2) 두 개의 정수를 받아 초기화하는 생성자를 작성하시오

```
public Fraction(int a, int b)
{
    numerator = a;
    denominator = b;
}
```

(3) 하나의 분수를 분자/분모 형태로 변환하는 ToString() 메소드를 작성하시오.

```
public override string ToString()
{
    return (numerator + "/" + denominator);
}
```

(4) 최대 공약수를 구하는 메소드와 기약 분수로 만드는 메소드를 작성하시오.

```

public void abbreviation()
{
    for(int i = 2; i < denominator; i++)
    {
        if(numerator % i == 0 && denominator % i == 0)
        {
            numerator = numerator / i;
            denominator = denominator / i;
            i = 2;
        }
    }
}

```

(5) 분수에 대한 4칙 연산을 수행하는 메소드 AddFraction, SubFraction, MulFraction, DivFraction을 작성하시오.

```

public Fraction AddFraction(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.numerator = f1.numerator * f2.denominator + f2.numerator * f1.denominator;
    result.denominator = f1.denominator * f2.denominator;
    result.abbreviation();
    return result;
}

```

```

public Fraction SubFraction(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.numerator = f1.numerator * f2.denominator - f2.numerator * f1.denominator;
    result.denominator = f1.denominator * f2.denominator;
    result.abbreviation();
    return result;
}

```

```

public Fraction MulFraction(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.numerator = f1.numerator * f2.numerator;
    result.denominator = f1.denominator * f2.denominator;
    result.abbreviation();
    return result;
}

```

```

public Fraction DivFraction(Fraction f1, Fraction f2)
{
    Fraction result = new Fraction();

    result.numerator = f1.numerator * f2.denominator;
    result.denominator = f1.denominator * f2.numerator;
    result.abbreviation();
    return result;
}

```

(6) 테스트 클래스를 만들어 테스트 하시오.

```

public Fraction()
{
    numerator = 0;
    denominator = 1;
}

```

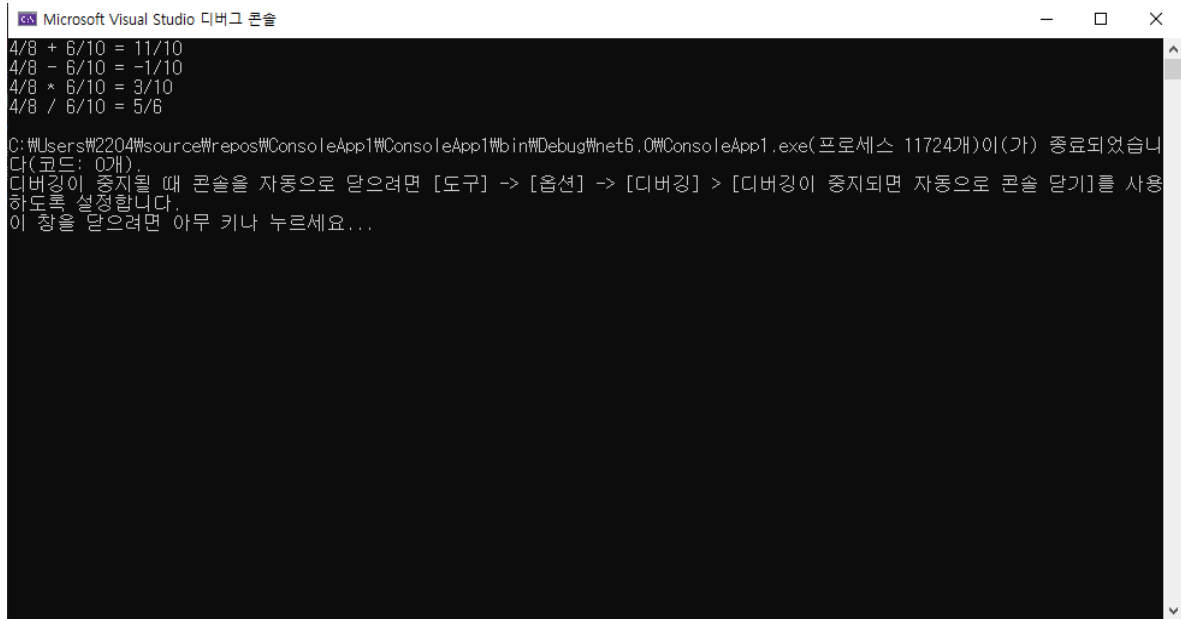
```

class FractionApp
{
    참조 0개
    public static void Main()
    {
        Fraction F1 = new Fraction(4,8);
        Fraction F2 = new Fraction(6,10);
        Fraction F3 = new Fraction();

        Fraction addf = F3.AddFraction(F1, F2);
        Fraction subf = F3.SubFraction(F1, F2);
        Fraction mutf = F3.MutFraction(F1, F2);
        Fraction divf = F3.DivFraction(F1, F2);

        Console.WriteLine(F1.ToString() + " + " + F2.ToString() + " = " + addf);
        Console.WriteLine(F1.ToString() + " - " + F2.ToString() + " = " + subf);
        Console.WriteLine(F1.ToString() + " * " + F2.ToString() + " = " + mutf);
        Console.WriteLine(F1.ToString() + " / " + F2.ToString() + " = " + divf);
    }
}

```



```
Microsoft Visual Studio 디버깅 콘솔
4/8 + 6/10 = 11/10
4/8 - 6/10 = -1/10
4/8 * 6/10 = 3/10
4/8 / 6/10 = 5/6

C:\Users\2204\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net6.0\ConsoleApp1.exe (프로세스 11724개)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>

- 한 개의 정수를 받는 생성자는 분자를 입력 받은 수, 분모를 1로 하여 생성하도록 작성하였고, 두 개의 정수를 받는 생성자는 분자를 첫번째 입력 받은 수, 분모를 두번째 입력 받은 수로 하여 생성하도록 작성하였다.
- ToString() 메소드는 Fraction 클래스를 분자 / 분모 형식의 string 객체로 리턴 하도록 작성하였다.
- abbreviation() 메소드는 최대공약수를 찾게 되면 Fraction 클래스 객체를 기약분수로 만들게 하는 메소드이다. `if(numerator % i == 0 && denominator % i == 0)` 문장이 분자와 분모의 최대공약수를 찾아 True가 된다면 분자와 분모를 i만큼 나누어 주게 한다.
- AddFraction 메소드는 Fraction 객체를 리턴하기 위한 result 객체에 F1 객체와 F2 객체의 분모를 서로의 수만큼 곱하고, 분자를 상대 객체의 분모의 수만큼 곱한 후 더한 값을 받는다. 그 후, abbreviation() 메소드를 실행시켜 기약분수로 바꾸고 result객체를 리턴한다.
- SubFraction 메소드도 마찬가지로 F1 객체와 F2 객체의 분자를 빼는 것 외에는 일치한다.
- MulFraction 메소드는 Fraction 객체를 리턴할 result 객체에 F1과 F2의 분모를 서로 곱한 값과 분자를 서로 곱한 값을 받은 뒤, 기약분수로 만들어 result 객체를 리턴한다.
- DivFraction 메소드는 Fraction 객체를 리턴할 result 객체에 F1의 분자와 F2의 분모를 곱한 값을 분자에, F1의 분모와 F2의 분자를 곱한 값을 분모에 받아 기약분수로 만들어 result 객체를 리턴한다.

- Main() 메소드에서는 F1과 F2의 Fraction 객체를 두 개 생성한 뒤, 값을 받지 않는 Fraction F3 객체를 생성하여 이 객체를 통해 F1과 F2의 덧셈, 뺄셈, 곱셈, 나눗셈의 결과를 받도록 한다. 결과 출력은 ToString() 메소드와 F3을 통해 생성한 addf, subf, mutf, divf 객체들을 통하여 출력하였다.

## <4.12>

(1) 한 개의 실수를 받아 초기화하는 생성자를 작성하시오.

```
public Complex(int a)
{
    real = a;
    image = 0;
}
```

(2) 두 개의 실수를 받아 초기화하는 생성자를 작성하시오.

```
public Complex(int a, int b)
{
    real = a;
    image = b;
}
```

(3) 하나의 복소수를 (실수부, 허수부) 형태로 변환하는 ToString() 메소드르 작성하시오.

```
public override string ToString()
{
    return (real + "+" + image + "i");
}
```

(4) 복소수에 대한 4칙 연산을 수행하는 메소드 AddComplex, MulComplex를 작성하시오.

```
public Complex AddComplex(Complex C1, Complex C2)
{
    Complex result = new Complex();
    result.real = C1.real + C2.real;
    result.image = C1.image + C2.image;
    return result;
}
```

```
public Complex MulComplex(Complex C1, Complex C2)
{
    Complex result = new Complex();
    result.real = (C1.real * C2.real) - (C1.image * C2.image);
    result.image = (C1.real * C2.image) + (C1.image * C2.real);
    return result;
}
```

(5) 테스트 클래스를 만들어 테스트 하시오.

```
public Complex()
{
    real = 0;
    image = 0;
}

class ComplexApp
{
    참조 0개
    public static void Main()
    {
        Complex C1 = new Complex(2, 3);
        Complex C2 = new Complex(4, 5);
        Complex C3 = new Complex();

        Complex AddResult = C3.AddComplex(C1, C2);
        Complex MutResult = C3.MulComplex(C1, C2);

        Console.WriteLine(C1.ToString() + " + " + C2.ToString() + " = " + AddResult.ToString());
        Console.WriteLine(C1.ToString() + " * " + C2.ToString() + " = " + MutResult.ToString());
    }
}
```



Microsoft Visual Studio 디버그 콘솔

2+3i + 4+5i = 6+8i  
2+3i \* 4+5i = -7+22i

C:\Users\2204\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net6.0\ConsoleApp1.exe(프로세스 8028개)이(가) 종료되었습니다(코드: 0개).  
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.  
이 창을 닫으려면 아무 키나 누르세요...

## <설명>

- 한 개의 실수를 받는 생성자는 실수부를 입력 받은 수로 설정하여 생성하고, 두 개의 정수를 받는 생성자는 첫번째 실수를 실수부로, 두번째 실수를 허수부로 설정하여 생성하도록 하였다.
- ToString() 메소드는 Complex 클래스를 실수부 + 허수부 형식의 string 객체로 리턴 하도록 작



성하였다.

- AddFraction 메소드는 Complex 객체를 리턴 하기 위한 매개변수가 없는 AddResult 객체에 C1 객체와 C2 객체의 실수부는 실수부끼리, 허수부는 허수부끼리 더하여 그 값을 AddResult 객체에 실수부와 허수부에 맞게 할당 후 리턴 한다.
- MutFraction 메소드는 Complex 객체를 리턴 하기 위한 매개변수가 없는 MutResult 객체를 생성 후, MutFraction 메소드의 매개변수인 Complex 클래스의 C1과 C2 객체의 복소수의 곱셈 공식 $\{(a + bi) * (c + di) = (ac - bd) + (ad + bc)i\}$ 을 이용하여 알맞게 실수부와 허수부에 할당한 후 리턴 한다.
- Main() 메소드에서는 C1과 C2의 Complex 객체를 두 개 생성한 뒤, 매개변수가 없는 Fraction C3 객체를 생성하여 이 객체를 통해 C1과 C2의 덧셈, 곱셈의 결과를 받도록 한다. C1(2, 3), C2(4, 5)의 덧셈은  $(2 + 4) + (3 + 5)i$  이므로  $6 + 8i$ 이다. 곱셈은  $\{(2 * 4) - (3 * 5)\} + \{(2 * 5) + (3 * 4)\}i$  이므로  $-7 + 22i$  이다. 결과 출력은 ToString() 메소드를 이용하여 C1, C2, AddResult, MutResult를 출력하였다.

## <4.13>

(1)디폴트 생성자를 작성하시오. 이 경우에 스택의 크기는 100이다.

```
public Stack()  
{  
    stack = new int[100];  
}
```

(2) 메소드 Push()와 Pop()을 작성하시오.

```
public void Push(int data)  
{  
    sp = sp + 1; stack[sp] = data;  
}
```

```
public int Pop()  
{  
    int tmp = stack[sp];  
    sp = sp - 1;  
    return tmp;  
}
```

(3) 클래스 Stack을 이용하여 일련의 정수 입력(입력의 끝은 0)을 역순으로 출력하는 C# 프로그램을 작성하시오.

```

class Stack
{
    private int[] stack;
    int sp = -1;
    참조 1개
    public Stack()
    {
        stack = new int[100];
    }
    참조 1개
    public Stack(int size)
    {
        stack = new int[size];
    }
    참조 2개
    public void Push(int data)
    {
        sp ++; stack[sp] = data;
    }
    참조 2개
    public int Pop()
    {
        int tmp = stack[sp];
        sp --;
        return tmp;
    }
    참조 4개
    public int returnLength()
    {
        return stack.Length;
    }
}

```

```


class StackApp
{
    참조 0개
    public static void Main()
    {
        Stack st1 = new Stack(10);
        Stack st2 = new Stack();

        Console.WriteLine("This Stack have 10 array");
        for (int i = 0; i < st1.returnLength(); i++)
        {
            st1.Push(i);
        }
        for(int j = 0; j < st1.returnLength(); j++)
        {
            Console.WriteLine(st1.Pop());
        }

        Console.WriteLine("This Stack is default stack");
        for (int i = 0; i < st2.returnLength(); i++)
        {
            st2.Push(i);
        }
        for (int j = 0; j < st2.returnLength(); j++)
        {
            Console.WriteLine(st2.Pop());
        }
    }
}

```

## <결과>

 Microsoft Visual Studio 디버그 콘솔

```
This Stack have 10 array
```

```
9  
8  
7  
6  
5  
4  
3  
2  
1  
0
```

```
This Stack is default stack
```

```
99  
98  
97  
96  
95  
94  
93  
92  
91  
90  
89  
88  
87  
86  
85  
84  
83  
82  
81  
80  
79  
78  
77  
76  
75  
74  
73  
72  
71  
70  
69  
68  
67  
66  
65  
64  
63  
62  
61  
60  
59  
58
```

```
Microsoft Visual Studio 디버그 콘솔

34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

C:\Users\2204\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\net6.0\ConsoleApp1.exe
(프로세스 932개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅
이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

## <설명>

- Stack 클래스는 정수형 배열을 가지고 있는 stack과 stack의 포인터 sp로 이루어져 있고, 디폴트 생성자는 100개의 정수형 배열을 가지도록, 매개변수가 있는 생성자는 매개변수만큼 정수형 배열을 가지도록 작성하였다.
- Push() 메소드는 sp변수를 1증가시킨 후 Push() 메소드의 매개변수를 증가된 sp가 가리키는 곳에 저장시킨다. Pop() 메소드는 sp가 가리키는 배열에 저장된 데이터를 tmp라는 변수에 임시로 저장시킨 후, sp를 1감소시킨 후 tmp변수를 리턴 시킨다. Push() 메소드에서 sp변수를 1증가시킨 후 데이터를 입력하는 이유는 sp가 배열의 장소가 아닌 곳 혹은 전에 저장된 데이터를 가리키고 있기 때문이다. Pop() 메소드에서는 먼저 sp가 가리키는 곳의 데이터를 리턴시켜야 하기 때문에 tmp 변수에 저장시킨 후, sp를 1감소시킨다.

- Main() 메소드에서 디폴트 stack과 매개변수를 가진 stack을 비교하기 위해 10개의 공간을 가진 st1과 디폴트 stack st2를 생성하였다. Stack에 스택의 길이는 구하는 returnLenghth() 메소드를 작성하였는데, 이 메소드는 스택의 길이를 리턴 시키도록 하였다. for문을 이용하여 returnLenghth()만큼 반복하여 for문의 변수인 i의 수를 st1, st2에 Push() 시킨 후 다시 for문을 이용하여 Pop 시켜 결과를 출력한다.

## <4.14>

(1) 크기를 받아 초기화하는 생성자를 작성하시오.

(2) 벡터 클래스를 테스트하는 프로그램을 작성하시오. 즉, 일련의 데이터(입력 데이터의 끝은 0)을 읽어 정렬한 후 출력하는 프로그램을 작성하는 것이다.

```
class Vector
{
    public int[] v;
    참조 0개
    public Vector()
    {
        v = new int[100];
    }

    참조 1개
    public Vector(int size)
    {
        v = new int[size];
    }

    참조 3개
    static void Swap(ref int x, ref int y)
    {
        int temp = x;
        x = y;
        y = temp;
    }
}
```

```

public void Qsort(int left, int right)
{
    int pe;
    int i, last;

    if (left >= right) return;
    pe = (left + right) / 2;
    Swap(ref v[left], ref v[pe]);
    last = left;
    for(i = left + 1; i <= right; i++)
    {
        if (v[i] < v[left])
        {
            Swap(ref v[++last], ref v[i]);
        }
    }
    Swap(ref v[left], ref v[last]);
    Qsort(left, last - 1);
    Qsort(last + 1, right);
}
참조 4개
public int returnLength()
{
    return v.Length;
}
}

```

```

class VectorApp
{
    참조 0개
    public static void Main()
    {
        Vector v1 = new Vector(10);

        Console.WriteLine("Before QSort :");

        for(int i = 0; i < v1.returnLength(); i++)
        {
            if (i == v1.returnLength() - 1)
            {
                v1.v[i] = 0;
                Console.WriteLine(v1.v[i]);
            }
            else
            {
                v1.v[i] = i * 7 % 10;
                Console.WriteLine(v1.v[i]);
            }
        }

        Console.WriteLine("After QSort :");
        v1.Qsort(0, v1.returnLength() - 1);

        for(int j = 0; j < v1.returnLength(); j++)
        {
            Console.WriteLine(v1.v[j]);
        }
    }
}

```

<결과>

```
Microsoft Visual Studio 디버그 콘솔
Before QSort :
0
7
4
1
8
5
2
9
6
0
After QSort :
0
0
1
2
4
5
6
7
8
9
```

## <설명>

- Vector 클래스는 데이터를 담을 배열 v가 있고, 디폴트 생성자는 100개의 정수를 담는 배열을 갖는다. 크기를 받아 초기화하는 생성자는 size 변수를 매개변수로 받아 그만큼의 길이를 갖는 배열을 갖는다.
- 메소드로는 Swap, Qsort, returnLength 등이 있는데, Swap() 메소드는 두 개의 변수의 순서를 서로 바꾸도록 ref 키워드를 사용하여 참조 호출을 발생시킨다. Qsort는 left 변수와 right 변수를 인자로 받는데, 먼저 pe라는 변수에 left와 right의 합의 절반값을 할당받고, Swap() 메소드를 이용하여 배열 v에 left와 pe의 위치를 서로 바꾼다. 그 후, last 라는 변수에 left 값을 받고, for문을 이용하여 left 위치에 있는 변수보다 작은 값이 있는 경우 그 수를 아까 받은 last에 ++last를 적용하고 last에 있는 데이터와 i에 있는 데이터를 바꾼다. 이런 식으로 바꾼 후, i가 right에 도달하면 left에 있는 데이터와 last에 있는 데이터를 바꾼 후 그 last를 기준으로 다시 Qsort를 진행하여 left가 right보다 크거나 같을 때까지 진행시킨다.
- Main() 메소드에서는 임의의 수를 입력할 수 있도록 for문의 변수에 7을 곱한 값에 % 10을 적용하여 0부터 9까지 수를 무작위로 입력하였다. 그 후 Qsort를 적용하여 Vector 클래스의 객체 v1을 정렬시키고 출력하였다.

## <4.17>

(1) 크기를 매개변수로 받아 초기화하는 생성자를 작성하시오.

```
public Vector(int size)
{
    v = new int[size];
}
```

(2) 정수형 인덱스를 매개변수로 받아 처리하는 인덱서를 작성하시오.

```
public int this[int index]
{
    get { return v[index]; }
    set { v[index] = value; }
}
```

(3) 벡터에 대한 ++와 -- 연산자를 중복하여 정의하시오. 벡터의 ++ 연산자는 배열에 속한 모든 원소를 1씩 증가하는 의미이고 -- 연산자는 1씩 감소하는 의미이다.

```
public static Vector operator ++(Vector v1)
{
    for(int i = 0; i < v1.v.Length; i++)
    {
        v1.v[i] += 1;
    }
    return v1;
}
```

참조 0개

```
public static Vector operator --(Vector v1)
{
    for (int i = 0; i < v1.v.Length; i++)
    {
        v1.v[i] -= 1;
    }
    return v1;
}
```

(4) 테스트 클래스를 만들어 테스트 하시오.



```

class Vector
{
    private int[] v;

    참조 2개
    public Vector(int size)
    {
        v = new int[size];
    }

    참조 8개
    public int this[int index]
    {
        get { return v[index]; }
        set { v[index] = value; }
    }

    참조 0개
    public static Vector operator ++(Vector v1)
    {
        for(int i = 0; i < v1.v.Length; i++)
        {
            v1.v[i] += 1;
        }
        return v1;
    }

    참조 0개
    public static Vector operator --(Vector v1)
    {
        for (int i = 0; i < v1.v.Length; i++)
        {
            v1.v[i] -= 1;
        }
        return v1;
    }
}

```

```

        return v1;
    }

    참조 4개
    public int returnLength()
    {
        return v.Length;
    }
}

참조 0개
class VectorApp
{
    참조 0개
    public static void Main()
    {
        Vector V1 = new Vector(5);
        Vector V2 = new Vector(5);

        Console.WriteLine("Before operator++ :");
        for (int i = 0; i < V1.returnLength(); i++)
        {
            V1[i] = i * 7 % 10;
            Console.WriteLine("V1[" + i + "] = " + V1[i]);
        }

        Console.WriteLine("After operator++ : ");
        for (int i = 0; i < V1.returnLength(); i++)
        {
            V1[i]++;
            Console.WriteLine("V1[" + i + "] = " + V1[i]);
        }
    }
}

```

```

        Console.WriteLine("Before operator-- : ");
        for (int i = 0; i < V2.returnLength(); i++)
        {
            V2[i] = i + 7 % 10;
            Console.WriteLine("V1[" + i + "] = " + V2[i]);
        }

        Console.WriteLine("After operator++ : ");
        for (int i = 0; i < V2.returnLength(); i++)
        {
            V2[i]--;
            Console.WriteLine("V1[" + i + "] = " + V2[i]);
        }
    }
}

```

## <결과>

```

Before operator++ :
V1[0] = 0
V1[1] = 7
V1[2] = 4
V1[3] = 1
V1[4] = 8
After operator++ :
V1[0] = 1
V1[1] = 8
V1[2] = 5
V1[3] = 2
V1[4] = 9

Before operator-- :
V1[0] = 0
V1[1] = 7
V1[2] = 4
V1[3] = 1
V1[4] = 8
After operator++ :
V1[0] = -1
V1[1] = 6
V1[2] = 3
V1[3] = 0
V1[4] = 7

```

## <설명>

- Vector 클래스는 정수형 배열을 받는 private 접근수정자를 가진 v와 크기를 매개변수로 받아 초기화하는 생성자, 정수형 인덱스를 매개변수로 받아 처리하는 인덱서, ++연산자와 --연산자를 중복 선언, v의 길이를 구해주는 returnLength() 메소드로 구성되어있다.
- 인덱서는 해당 인덱스에 해당하는 값을 리턴해주는 get 프로퍼티와 value를 입력받을 때 해당 인덱스에 값을 할당하는 set 프로퍼티로 이루어져 있다.
- ++연산자와 --연산자의 중복 선언은 v배열의 처음부터 끝까지 for문을 이용하여 해당 인덱스

에 해당되는 데이터들을 +1하거나 -1을 시켜준다.

- `returnLength()` 메소드는 `v`배열이 `private`로 선언되어 있기에 길이를 리턴시켜 주기 위해 작성하였다.
- `Main()` 메소드에서는 ++연산자와 --연산자의 중복을 따로 알아보기 위해 `V1`, `V2`를 선언하였고 크기는 5로 지정하였다. `for` 문을 이용하여 각 `Vector` 객체에 `i * 7 % 10`에 해당되는 값들을 넣어주는 `set` 프로퍼티가 작동되었고, `get` 프로퍼티를 작동시켜 `Console.WriteLine` 문에서 출력되도록 하였다. `V1`과 `V2`는 각각 `for`문 안에서 ++연산자와 --연산자를 적용시켜 각 인덱스에 해당되는 값들을 +1, -1 하였고, 전의 값들과 비교하기 위해 전 값과 연산자 적용 후의 값을 따로 출력하였다.