

본 강의에서 수업자료로 이용되는 저작물은
저작권법 제25조 수업목적 저작물 이용 보상금제도에 의거,
한국복제전송저작권협회와 약정을 체결하고 적법하게 이용하고 있습니다.
약정범위를 초과하는 사용은 저작권법에 저촉될 수 있으므로
수업자료의 재 복제, 대중 공개·공유 및 수업 목적 외의 사용을 금지합니다.

2024. 8. 30.

부천대학교·한국복제전송저작권협회

C#

2주차 1교시

2장 언어 구조 2 와 3장 문장 종류

2주차 학습 내용

- * 1교시

- * 2장 C# 언어 구조 2

- * 자료형(값형:열거형, 참조형: 배열)
 - * 형 변환
 - * 박싱과 언박싱

- * 3장 문장 종류

- * 제어문

- * 2교시

- * 8장 컨트롤

- * 윈폼 애플리케이션 작성 순서(7장 참조)
 - * 버튼 기반 컨트롤(버튼, 라디오 버튼)

- * 3교시

- * 8장 컨트롤

- * 버튼 기반 컨트롤(체크 상자)
 - * 레이블과 링크 레이블
 - * 텍스트 상자

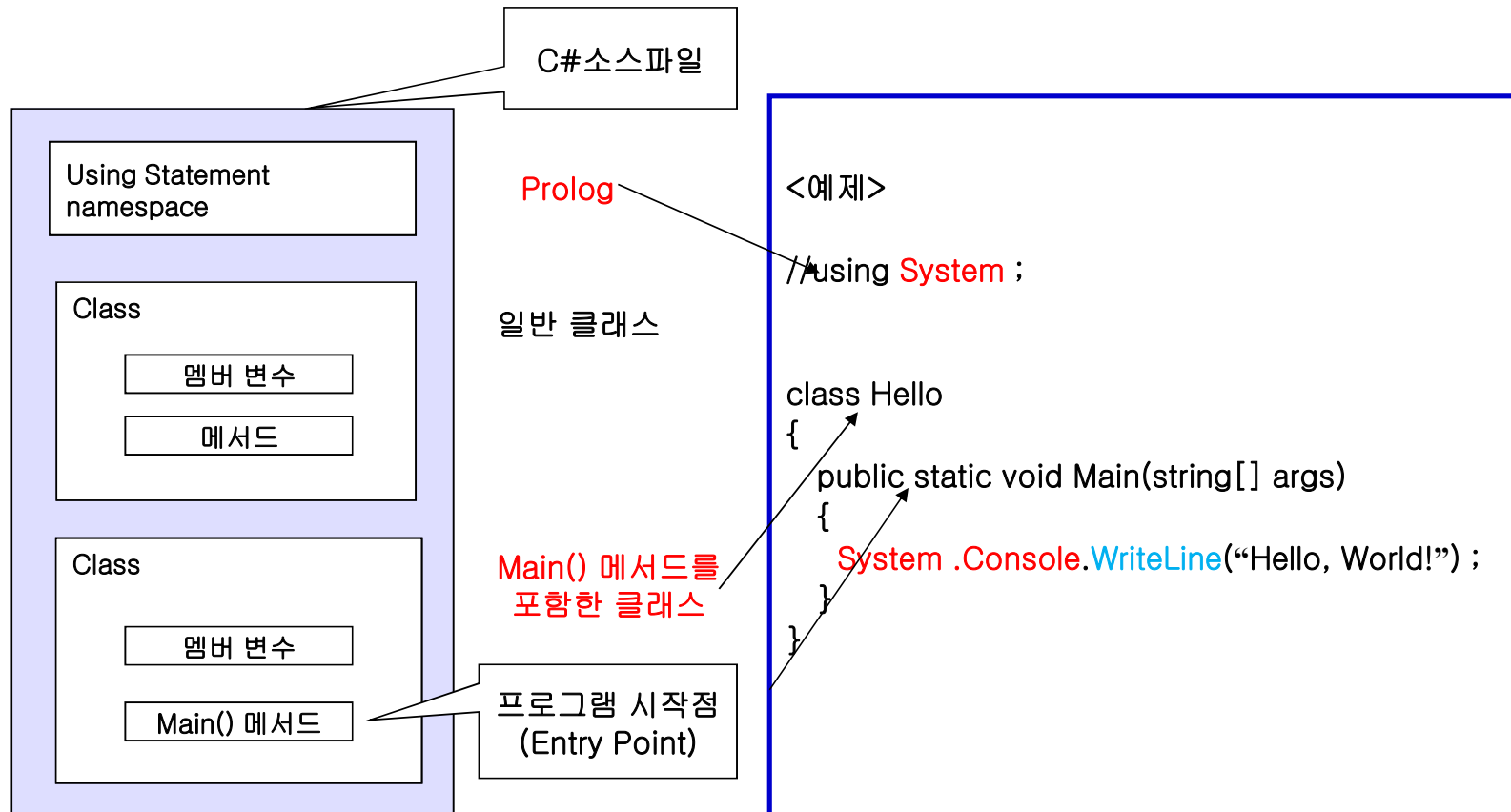
C# 프로그래밍



1. 개요

1장 C# 프로그램 구조

◆ C# 프로그램 구조



C# 프로그래밍



2. 언어 구조 2

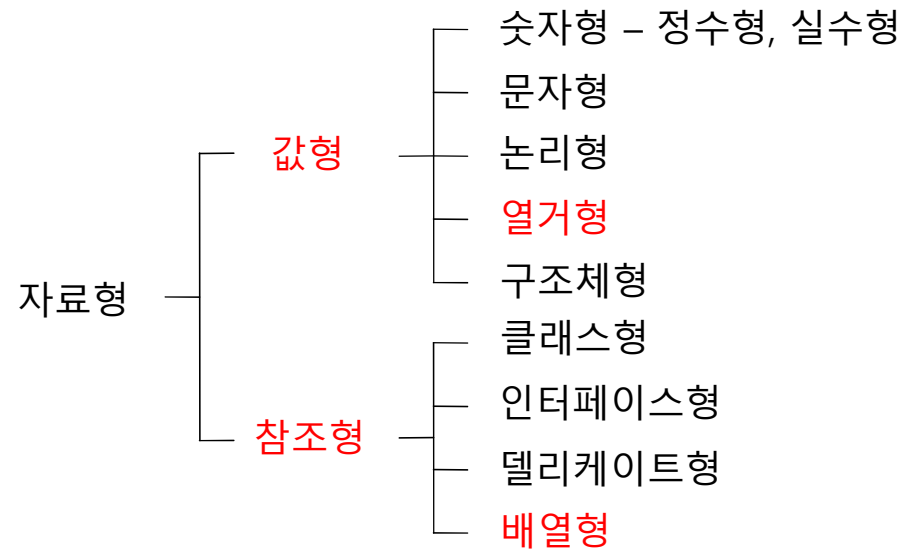


자료형 [1/2]

■ 자료형의 의미

- 자료 객체가 갖는 형으로 구조 및 개념, 값의 범위, 연산 등을 정의

■ 자료형의 종류(값을 저장하는 방법에 따라)





자료형 [2/2]

- C# 의 자료형은 공통자료형 시스템(CTS)에서 정의한 형식으로 표현할 수 있다.

// 다음 두 선언의 의미는 동일하다.

System.Int32 x; // CTS 형으로 정수형 변수 x의 선언

int x; // C# 형으로 정수형 변수 x의 선언

- CTS 형과 C# 자료형과의 관계

CTS 자료형	의미	C#자료형	CTS 자료형	의미	C#자료형
System.Object	객체형	object	System.Int64	64비트 정수형	long
System.String	스트링형	string	System.UInt64	64비트 부호없는 정수형	ulong
System.Sbyte	부호있는 바이트형	sbyte	System.Char	문자형	char
System.Byte	바이트형	byte	System.Single	단일 정밀도 실수형	float
System.Int16	16비트 정수형	short	System.Double	이중 정밀도 실수형	double
System.UInt16	16비트 부호없는 정수형	ushort	System.Boolean	불린형	bool
System.Int32	32비트 정수형	int	System.Decimal	10진수형	decimal
System.UInt32	32비트 부호없는 정수형	uint			



값형 - 열거형

- 열거형의 의미
 - 서로 관련 있는 상수들의 모음을 심볼릭한 명칭의 집합으로 정의한 것
- 기호상수
 - 집합의 원소로 기술된 명칭
- 순서값
 - 집합에 명시된 순서에 따라 0부터 부여된 값
 - 정수형으로 교환하여 사용할 수 있다.
- 변수가 가질 수 있는 값이 특정한 유한 집합으로 제한할 때 사용
- 열거형 멤버 자체를 열거타입.멤버 식으로 사용



값형 - 열거형

예제 2.9 [ch2_ex2_9_xxxx]

```
using System;
enum Color {Red, Green, Blue} //
class EnumTypeApp
{
    public static void Main()
    {
        Color c = Color.Red; //
        c++;
        int i = (int) c;
        Console.WriteLine("Cardinality of " + c + " = " + i);
    }
}
```

실행 결과 :

Cardinality of Green =1



값형 - 열거형

퀴즈 2.2 [ch2_quiz2_xxxx] 다음 프로그램의 실행 결과 값을 표현 하시오.

```
using System;
enum SignFlag {black, yellow=3, green, blue=7, red}

class EnumTest2{
    public static void Main(){
        SignFlag s1;
        s1 = SignFlag.black;
        Console.WriteLine("{0}={1}", s1, (int)s1);
        s1 = SignFlag.yellow;
        Console.WriteLine("{0}={1}", s1, (int)s1);

        Console.WriteLine("{0}={1}", SignFlag.green, (int)SignFlag.green);
        Console.WriteLine("{0}={1}", SignFlag.blue, (int)SignFlag.blue);
        Console.WriteLine("{0}={1}", SignFlag.red, (int)SignFlag.red);
    }
}
```

실행 결과 :



참조형 - 배열형

■ 배열형의 의미

- 같은 형의 여러 개의 값을 저장하는데 사용하는 자료형
- 순서가 있는 원소들의 모임

■ 배열의 종류

- 형태별 분류
 - 사각형 배열, 가변형 배열(배열의 배열), 혼합형 배열
- 차원에 의한 분류
 - 단순 배열(1차원 배열), 다차원 배열



참조형 - 배열형

■ 배열을 사용하기 위한 과정

■ 배열 선언

- 배열이름, 차원, 그리고 원소의 형 등을 명시


```
int[]    vector;           // 1차원 배열(단순 배열)
short[,] matrix;          // 2차원 배열(사각형 배열)
object[] myArray;
int[]    initArray = {0, 1, 2, 3, 4, 5}; // 선언과 함께 초기값 부여
int[][]  arrayOfArray;    // 2차원 배열(가변형 배열)
```

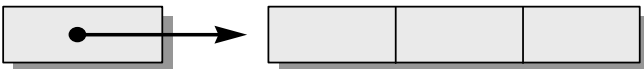
■ 배열 객체 생성

- new 연산자를 통해서 동적으로 생성
- 배열의 객체를 생성함으로써 배열 이름은 특정 배열 객체를 가리킴

```
vector = new int[100];
matrix  = new short[10,100];
myArray = new Point[3];
```

- 배열 객체 선언과 생성

int[] vector ; vector : 

vector = new int[3] ; vector : 
vector[0] vector[1] vector[2]

■ 배열에 값 저장

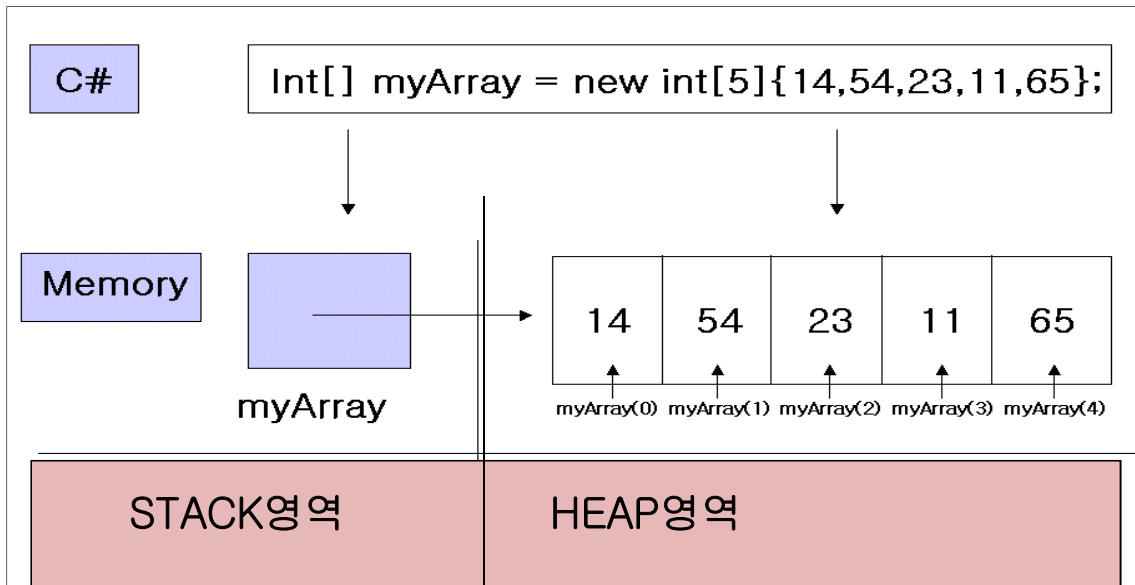
- 배열의 인덱스는 0부터 시작
- Length 프로퍼티 통한 배열의 길이 접근
- 인덱스 범위 초과 : IndexOutOfRangeException 발생

```
int[] vector = new int[100];
// ...
for (int i=0; i < vector.Length; i++) vector[i] = i;
```



단순 배열

■ 배열의 초기화 후의 모습



int형의 5개의 요소를 가진 배열 myArray가 정상적으로 초기화가 되면, 메모리에 배열이 저장되는데, 배열명은 STACK 이라는 곳에, 각 요소들의 값은 HEAP 이라는 곳에 저장이 된다

■ 배열 선언 시 주의 사항

- 선언 시 배열 변수 크기를 지정 할 수 없음

```
int[10] arr;           // 에러 처리
```



사각형 배열

■ 배열의 선언과 배열 생성을 결합한 형태

```
데이터타입 [ , ... , ] 변수이름;  
변수이름 = 배열선언;
```

```
데이터타입[] 변수이름 = new 배열선언;
```

■ 배열 생성의 결합 형태 예

```
int [ , ] arr;  
arr = new int[3,4];
```

또는

```
int [ , ] arr=new int[3,4];
```

	↓ 0열	↓ 1열	↓ 2열	↓ 3열
0행 →	arr[0,0]			
1행 →			arr[1,2]	
2행 →		arr[2,1]		arr[2,3]

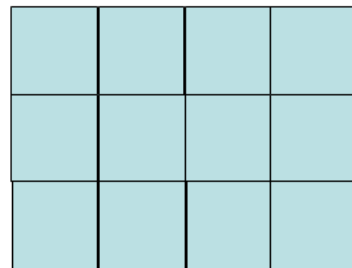


가변형 배열(배열의 배열)

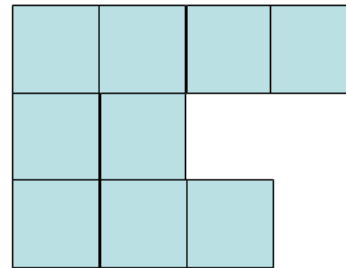
■ 가변형 배열

- 배열들이 모여서 이루어진 불규칙적인 배열
- 울퉁불퉁 배열, 다단계 배열, 배열의 배열이라고도 함
- 사각형 배열과 가변형 배열의 비교

사각형 배열



가변형 배열





가변형 배열

가변형 배열에서의 선언과 배열 생성 결합

데이터타입 `[]...[]` 변수이름;
변수이름=배열선언;

데이터타입 `[]...[]` 변수이름=배열선언;

배열 생성의 결합 형태 예

```
int[ ][ ] arr=new int[3][ ];  
arr[0] = new int[3];   arr[1] = new int[2];   arr[2] = new int[4]
```

1
`int[] arr1= {3,1,5};`
`int[] arr2={6,9};`
`int[] arr3={7,8,4,6};`
`int[][] arr_sawlike={arr1,arr2,arr3};`

2
`int[][] arr_sawlike=new int[3][]`
`{new int[]{3,1,5},`
`new int[]{6,9},`
`new int[]{7,8,4,6}};`

3
`int[][] arr_sawlike=new int[][]`
`{new int[]{3,1,5},`
`new int[]{6,9},`
`new int[]{7,8,4,6}};`

4
`int[][] arr_sawlike={new int[]{3,1,5},`
`new int[]{6,9},`
`new int[]{7,8,4,6}};`

3	1	5	
6	9		
7	8	4	6

가변형 배열 사용의 주의

`int [][] arr={{3,1,5},{6,9},{7,8,4,6}}` // 값의 입력 시 **new** 사용 해야 함(불가)



참조형 - 배열형

예제 2.10 [ch2_ex2_10_xxxx]

```
using System;
class ArrayTypeApp
{
    public static void Main()
    {
        int[] ia = new int[3];
        int[] ib = {1, 2, 3};
        // int[] ib = new int[] {1, 2, 3} 같은 의미이다
        int i;
        for (i = 0; i < ia.Length; i++)
            ia[i] = ib[i];
        for (i = 0; i < ia.Length; i++)
            Console.Write(ia[i] + " ");
        Console.WriteLine();
    }
}
```

실행 결과 :

1 2 3



참조형 - 배열형

예제 2.11 [ch2_ex2_11_xxxx]

```
using System;
class ArrayOfArrayApp {
    public static void Main() {
        int[][] arrayOfArray = new int[3][];           // declaration
        int i, j;
        for (i = 0; i < arrayOfArray.Length; i++)      // creation
            arrayOfArray[i] = new int[i+3];
        for (i = 0; i < arrayOfArray.Length; i++)      // using
            for (j = 0; j < arrayOfArray[i].Length; j++)
                arrayOfArray[i][j] = i*arrayOfArray[i].Length + j;
        for (i = 0; i < arrayOfArray.Length; i++) {    // printing
            for (j = 0; j < arrayOfArray[i].Length; j++)
                Console.Write(" " + arrayOfArray[i][j]);
            Console.WriteLine();
        }
    }
}
```

실행 결과 :

0	1	2		
4	5	6	7	
10	11	12	13	14



값과 참조

- 값 타입과 참조 타입의 차이점
 - 데이터가 저장되는 메모리상의 위치
 - 값 타입: 크기가 작고 고정적이기 때문에 **스택(stack)**에 생성
 - 참조 타입 : 크고 가변적이기 때문에 동적으로 관리 되는 **힙(heap)**에 생성
 - 데이터 사용 시점
 - 값 타입 : 선언만 하면 **스택에 즉시 생성되므로** 선언 직후부터 데이터를 저장하는 용도로 사용가능
 - 참조 타입: 선언에 의해 참조만 생성되므로 **new연산자로 메모리를 할당 받아** 초기화해야 사용 가능
 - 변수를 비교하는 방법
 - 값 타입: 기억된 데이터만 같으면 두 변수를 같은 것으로 평가
 - 참조 타입 : 힙상의 번지를 비교하기 때문에 내용이 완전히 같더라도 위치가 다르면 같지 않은 것으로 평가
- 값형 : 스택영역, 변수 값 자체를 가짐 ,**크기가 작을 때, 즉시(빠르게) 사용**, 데이터 값 비교
=> **데이터에 접근이 빠름.**
- 참조형 : 힙영역, 값이 저장된 위치만을 가짐,**크기가 크고 가변적일 때 사용**, 주소 비교
=> **데이터의 관리면에서 효율적임.**



값과 참조

퀴즈 2.3 [ch2_quiz3_xxxx] 다음 프로그램의 실행 결과 값을 표현 하시오.

```
using System;

class CompareValue
{
    static void Main()
    {
        int value = 3, vcopy = 3;
        Console.WriteLine("값 타입의 경우 : " + (value == vcopy ? "같다" : "다르다"));

        int[] ar = { 1, 2, 3, 4, 5 };
        int[] arcopy = { 1, 2, 3, 4, 5 };
        Console.WriteLine("참조 타입의 경우 : " + (ar == arcopy ? "같다" : "다르다"));
    }
}
```

실행 결과 :



연산자

■ 연산자 종류

C# 언어의 연산자

- 산술 연산자 : + - * / % 단항+ 단항-
- 관계 연산자 : > >= < <= == !=
- 논리 연산자 : && || !
- 증감 연산자 : 전위++ 전위-- 후위++ 후위--
- 비트 연산자 : & | ^ ~ << >>
- 조건 연산자 : ? :
- 배정 연산자 : = += -= *= /= %= &= |= ^= <<= >>=
- 캐스트 연산자 : (자료형)
- 형 검사 연산자 : is as (C#에서 새롭게 추가된 연산자)
- 배열 연산자 : []
- 메소드 연산자 : ()
- 멤버 접근 연산자 : .
- 지정어 연산자 : new typeof checked unchecked

■ 형 검사 연산자(type testing operator)란 C#에서 새롭게 추가된 연산자로 is, as 연산자가 있다.

■ 연산자 종류

■ 데이터 타입이 지정한 타입과 호환 가능한지 검사: is

```
obj is <type>
```

■ 주어진 값을 지정한 타입으로 변환: as

```
obj as <type>
```



연산자	결합법칙	우선순위
() [] . 후위++ 후위-- new typeof checked unchecked	좌측결합	<div style="display: flex; align-items: center; justify-content: center;"> <div style="flex: 1; border-left: 2px solid #0072bc; margin: 0 10px; position: relative;"> <div style="position: absolute; top: -10px; right: -10px;">↑</div> <div style="position: absolute; bottom: -10px; right: -10px;">↓</div> </div> <div style="text-align: center; flex: 1;"> <p>(높음)</p> <p>(낮음)</p> </div> </div>
단항+ 단항- ! ~ 전위++ 전위-- (자료형)	우측결합	
* / %	좌측결합	
+ -	좌측결합	
<< >>	좌측결합	
< > <= >= is as	좌측결합	
== !=	좌측결합	
&	좌측결합	
^	좌측결합	
	좌측결합	
&&	좌측결합	
	좌측결합	
? :	우측결합	
= += -= *= /= %= &= ^= = <<= >>=	우측결합	



형 변환

- 묵시적 형 변환(implicit type conversion)
 - 컴파일러에 의해 자동적으로 수행되는 형 변환
 - 작은 크기 자료형 → 큰 크기 자료형
- 명시적 형 변환(explicit type conversion)
 - 프로그래머가 **캐스트 연산자**를 사용하여 수행하는 형 변환
 - 형태
 - (자료형) 식
 - 큰 크기 자료형에서 작은 크기 자료형으로 변환 시 정밀도 상실
- 형 변환 금지(cast 연산자 사용할 때)
 - bool
 - 같은 자료형 이외에 다른 자료형으로의 변환 금지
 - **논리형을 숫자로 캐스트연산자로 변환 할 수가 없다.**

```
(int) 3.75    ==> 3
(float) 3     ==> 3.0
(float) (1 / 2) ==> 0.0
(float) 1 / 2 ==> 0.5
```

```
int i = 0xffff;
short s;
s = (short)i;
```




형 변환

■ 캐스트 연산자 : (자료형) => 값 타입 변환

예)

- 정수형과 더블형 사이의 변환 : 가능
- 열거형과 정수형 사이의 변환 : 가능
- double과 decimal 사이의 변환 : 가능
 - Decimal : 10진 자료형, 회계 금융에서 사용, 고도의 정밀도(유효숫자는 28자릿수)
- (문자열형, 논리형)과 정수형 사이의 변환 : 불가능

■ (문자열형, 논리형)과 정수형 사이의 변환 가능한 방법

■ Convert.ToXXX메서드(Convert클래스)이용

: 문자열을 수치(정수, 더블)으로 변환, 논리형을 정수형으로 변환

```
int i=Convert.ToInt32("1234");
```

■ Parse메서드 이용 : 문자열을 해당 타입으로 변환

```
int i=int.Parse("1234"); float f=float.Parse("67.12");
```

■ ToString메서드 이용 : 모든 타입을 문자열로 변환

```
int i=1234; string s=i.ToString();
```

④ float float.Parse(string s) (+ 3 오버로드)
숫자의 문자열 표현을 해당하는 단정밀도 부동 소수점 숫자로 변환

예외:
ArgumentNullException
FormatException
OverflowException



박싱과 언박싱

- C#에서는 값형과 참조형의 변환을 용이하게 하면서 시스템의 단일화를 이루려 한다.
 - 값형은 신속하게 그 데이터에 접근할 수 있는 반면, 참조형은 데이터의 관리 면에 있어 효율적인 장점이 있다. 하지만 값형을 참조형으로 또는 그 반대로 사용해야 하는 경우가 발생한다.

- 박싱(boxing)

- 값형의 데이터를 참조형으로 변환하는 것
- 컴파일러에 의해 묵시적으로 행해짐
- 스택(stack)에 있는 데이터가 힙(heap)으로 복사됨
- 박싱 과정

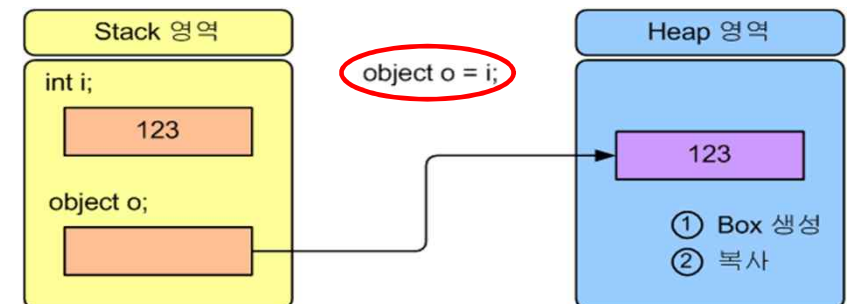
<박싱 예제>

```
int i = 123 ;
```

```
Object o = i ;
```

- 언박싱(unboxing)

- 참조형의 데이터를 값형으로 변환하는 것
- 반드시 캐스팅을 통하여 명시적으로 행해짐
- 반드시 박싱될 때 형으로 언박싱을 해주어야 함
- 힙에 있는 데이터가 스택으로 복사됨

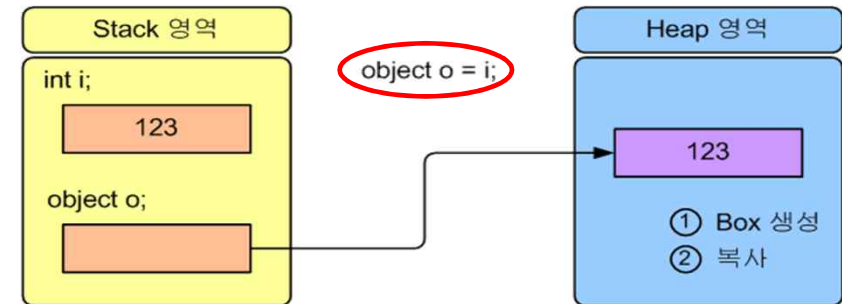




박싱과 언박싱

■ 박싱과 언박싱

- 값형->참조형 : 박싱(boxing) => 암시적
- 참조형->값형 : 언박싱(unboxing) => 명시적



박싱(boxing)
정수형(값형) 변수 i를 참조형 변수 o에
할당하게 되면 **암시적으로** i는 참조형으
로 형변환되게 된다.

언박싱(unboxing)
정수형 변수 y에 참조형 변수를 할당하려
고 하면 **명시적으로 캐스팅해서** 참조형을
정수형으로 변환하게 된다.

<예제>

```
int i = 123 ;
```

```
object o = i ;
```

```
int y = (int) o ;
```



형 변환 퀴즈

퀴즈 2.4 [ch2_quiz4_xxxx] 다음 프로그램의 실행 결과 값을 표현 하시오.

```
using System;

namespace Es02_12_01
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            float b, c ;

            Console.Write("Enter a : ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Enter b : ");
            b = float.Parse(Console.ReadLine());
            c = a + b ;
            Console.WriteLine("a = {0}, b = {1}, c = {2} ", a, b, c);
        }
    }
}
```

실행 결과

⊗ float float.Parse(string s) (+ 3 오버로드)

숫자의 문자열 표현을 해당하는 단정밀도 부동 소수점 숫자로 변환

예외:

ArgumentNullException

FormatException

OverflowException

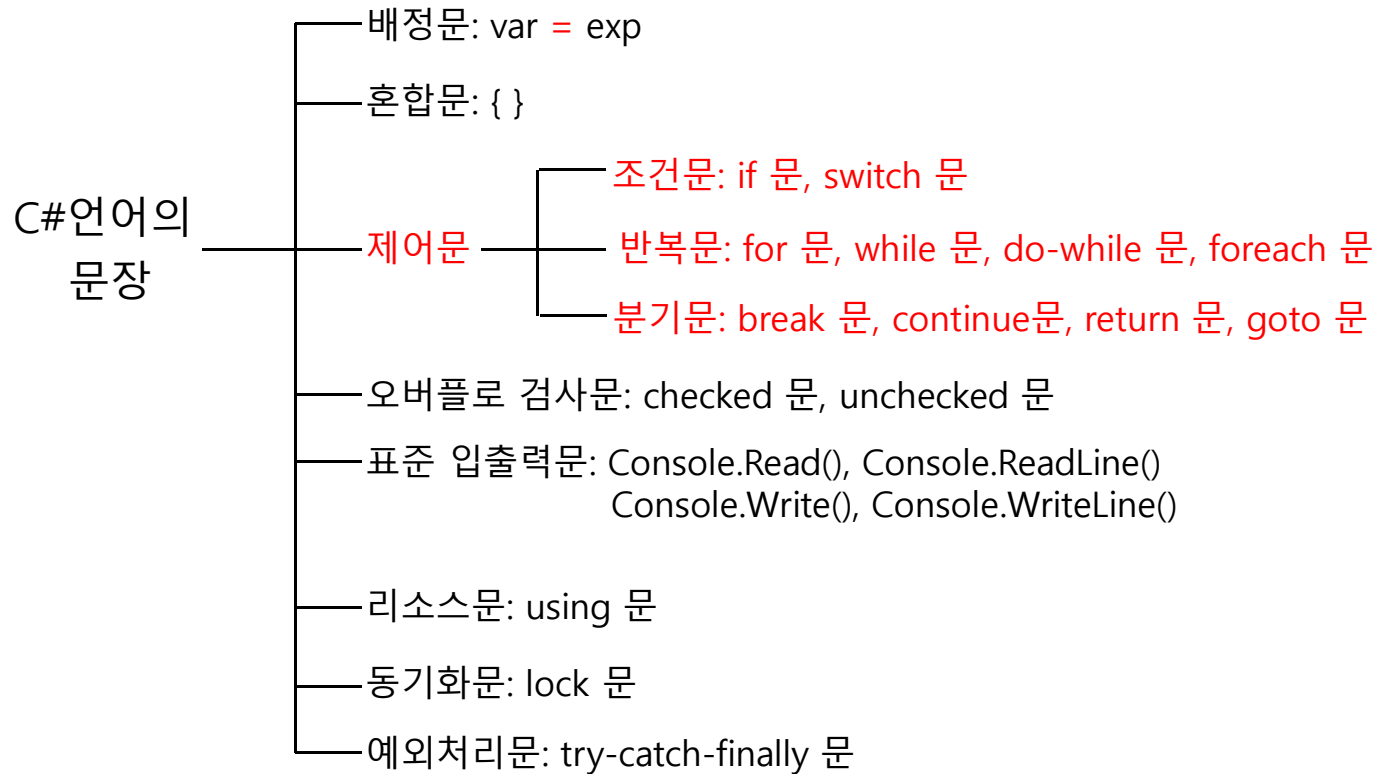
C# 프로그래밍 입문



3. 문장



문장의 종류





제어문

- 프로그램의 실행 순서를 바꾸는 데 사용
- 실행 순서를 제어하는 방법에 따라
 - 조건문 : if 문, switch 문
 - 반복문 : for 문, while 문, do-while 문, foreach 문
 - 분기문 : break 문, continue 문, return 문, goto 문



조건문 - if 문

- 조건에 따라 실행되는 부분이 다를 때 사용
- if 문 형태

```
if ( <조건식> ) <문장>  
if ( <조건식> ) <문장1> else <문장2>
```

- 조건식의 연산결과 : **논리형 (true or false)**

- 예

```
if (a < 0) a = -a;           // 절대값  
if (a > b) m = a; else m = b; // 큰값
```



조건문 - if 문

■ 내포된 if 문

- 참 부분에서 if 문이 반복

```
if(<조건식>
  if(<조건식>
    // ...
    <문장>
```

- else 부분에서 if 문이 반복

```
if(<조건식1>) <문장1>
else if(<조건식2>) <문장2>
...
else if(<조건식n>) <문장n>
else <문장>
```



조건문 - switch 문

- 조건에 따라 여러 경우로 처리해야 되는 경우
- switch 문의 형태

```
switch (<식>) {  
    case <상수식1> : <문장1> break;  
    case <상수식2> : <문장2> break;  
    .  
    .  
    case <상수식n> : <문장n> break;  
    default : <문장> break;  
}
```

- 제어변수: 정수, 열거형, 문자형, **문자열**
- default의 의미는 otherwise
- break 문을 사용하여 탈출, break문 생략 불가!
- case문 자체를 비워 둘 수는 있음
- case의 끝에 break대신 goto문을 사용할 수 있음!



조건문 - switch 문

[예제 3.9 – SwitchStWithStringApp.cs]

```
using System;
class SwitchStWithStringApp
{
    public static void Main()
    {
        Console.Write("Enter the weekday (Sunday-Saturday) : ");
        string day = Console.ReadLine();
        switch (day)
        {
            case "Sunday": Console.WriteLine(1); break;
            case "Monday": Console.WriteLine(2); break;
            case "Tuesday": Console.WriteLine(3); break;
            case "Wednesday": Console.WriteLine(4); break;
            case "Thursday": Console.WriteLine(5); break;
            case "Friday": Console.WriteLine(6); break;
            case "Saturday": Console.WriteLine(7); break;
            default: Console.WriteLine("Illegal day"); break;
        }
    }
}
```

입력 데이터:

Enter the weekday (Sunday-Saturday) : Sunday

실행 결과 :



조건문 - switch 문

[예제 3.9 – SwitchStWithStringApp.cs]

```
using System;
class SwitchStWithStringApp
{
    public static void Main()
    {
        Console.Write("Enter the weekday (Sunday-Saturday) : ");
        string day = Console.ReadLine();
        switch (day)
        {
            case "Sunday": Console.WriteLine(1); break;
            case "Monday": Console.WriteLine(2); break;
            case "Tuesday": Console.WriteLine(3); break;
            case "Wednesday": Console.WriteLine(4); break;
            case "Thursday": Console.WriteLine(5); break;
            case "Friday": Console.WriteLine(6); break;
            case "Saturday": Console.WriteLine(7); break;
            default: Console.WriteLine("Illegal day"); break;
        }
    }
}
```

// C#에서는 C나 C++와 다르게 case문의 상수 식 부분에 결과 값이 정수뿐만 아니라 **스트링도 가능**
입력 데이터:

Enter the weekday (Sunday-Saturday) : Sunday

실행 결과 :

1



반복문 - for 문

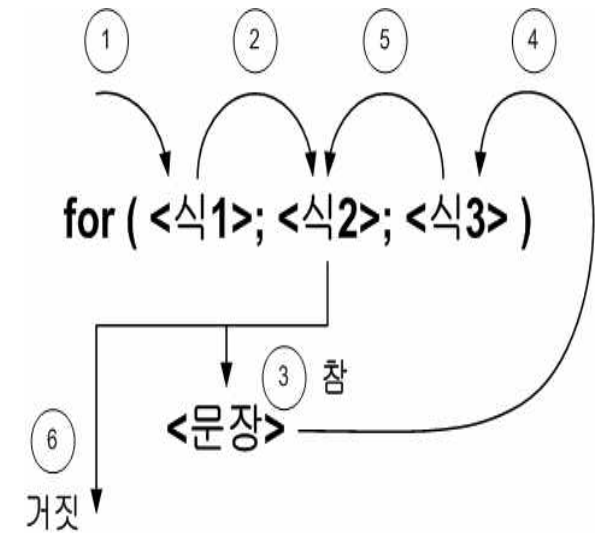
- 정해진 횟수만큼 일련의 문장을 반복
- for 문의 형태

```
for ( <식1> ; <식2> ; <식3> )  
    <문장>
```

- <식1> : 제어 변수 초기화
- <식2> : 제어 변수를 검사하는 조건식
- <식3> : 제어 변수의 값을 수정

예

```
s = 0;  
for (i = 1; i <= N; ++i) // 1부터 N까지의 합 : i 증가  
    s += i;
```





반복문 - while 문

■ while 문의 형태

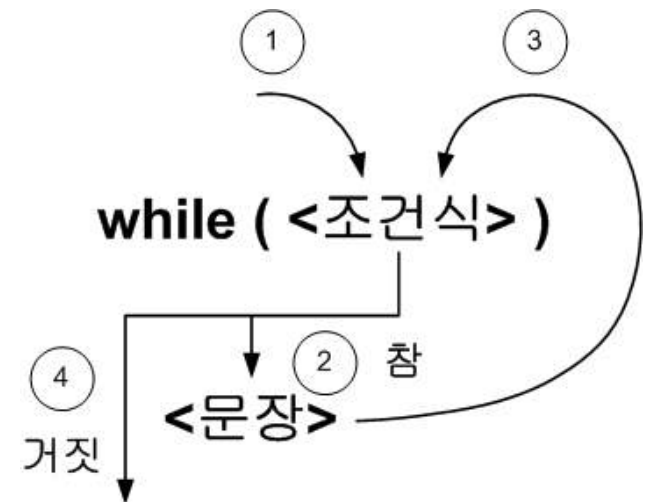
```
while ( 조건식 )  
    <문장>
```

■ 예

```
i = 1; s = 0;  
while (i <= N) { // 1부터 N까지의 합  
    s += i;  
    ++i;  
}
```

■ do ~ while 문의 형태

```
do  
    <문장>  
while ( <조건식> );
```





반복문 - foreach 문

- 데이터의 집합에 대한 반복을 수행
- foreach 문의 형태

```
foreach ( 자료형 변수명 in 데이터의 집합 )  
    <문장>
```

- 예

```
foreach ( string s in color )  
    Console.WriteLine(s);
```



```
for (int i=0; i<color.Length; i++)  
    Console.WriteLine(color[i]);
```



반복문 - foreach 문

[예제 3.15 - ForeachStApp.cs]

```
using System;
class ForeachStApp
{
    public static void Main()
    {
        string[] color = {"red", "green", "blue"};
        foreach (string s in color)
            Console.WriteLine(s);
    }
}
```

실행 결과 :

red
green
blue



분기문 - break 문

- 블록 밖으로 제어를 옮기는 역할
- break 문의 형태

```
break;
```

- 예

```
int i = 1;  
while (true) {  
    if (i == 3)  
        break;  
    Console.WriteLine("This is a " + i + " iteration");  
    ++i;  
}
```



분기문 - continue 문

- 다음 반복이 시작되는 곳으로 제어를 옮기는 기능
- continue 문의 형태

```
continue;
```

- for 문 안에서 사용될 때

```
for(i = 0; i <= 5; ++i) {  
    if (i % 2 == 0)  
        continue;  
    Console.WriteLine("This is a " + i + " iteration");  
}
```

2주차 학습 내용 정리

* 1교시

* 2장 C# 언어 구조 2

- * 자료형(값형:열거형, 참조형: 배열)
- * 형 변환
- * 박싱과 언박싱

* 3장 문장 종류

- * 제어문

* 2교시

* 8장 컨트롤

- * 윈폼 애플리케이션 작성 순서(7장 참조)
- * 버튼 기반 컨트롤(버튼, 라디오 버튼)

* 3교시

* 8장 컨트롤

- * 버튼 기반 컨트롤(체크 상자)
- * 레이블과 링크 레이블
- * 텍스트 상자