

제 3 장 문장

- **배정문**
- **혼합문**
- **제어문**
- **오버플로 검사문**
- **표준 입출력문**

문장의 종류

C#언어의 문장

배정문: `var = exp`

혼합문: `{ }`

제어문

조건문: `if` 문, `switch` 문

반복문: `for` 문, `while` 문, `do-while` 문, `foreach` 문

분기문: `break` 문, `continue`문, `return` 문, `goto` 문

오버플로 검사문: `checked` 문, `unchecked` 문

표준 입출력문: `Console.Read()`, `Console.ReadLine()`
`Console.Write()`, `Console.WriteLine()`

리소스문: `using` 문

동기화문: `lock` 문

예외처리문: `try-catch-finally` 문

배정문

- 값을 변수에 저장하는데 사용
- 형태 : <변수> = <식>;

```
remainder = dividend % divisor;  
i = j = k = 0;  
var op= exp;
```

- 형 변환
 - 묵시적 형 변환 : 컴파일러에 의해 자동
 - 명시적 형 변환 : 프로그래머가 캐스트(cast) 연산자

혼합문

- 여러 문장을 한데 묶어 하나의 문장으로 나타냄
 - 주로 문장의 범위를 표시
- 형태: { <선언> 또는 <문장> }

`if (a > b) a--; b++;`

`if (a > b) { a--; b++; }`
- 지역변수(Local Variable)
 - 블록의 내부에서 선언된 변수
 - 선언된 블록 안에서만 참조 가능

[예제 3.4 – LocalVariableApp.csc]

```
using System;
class LocalVariableApp {
    static int x;
    public static void Main() {
        int x = (LocalVariableApp.x=2) * 2;
        Console.WriteLine("static x = " + LocalVariableApp.x);
        Console.WriteLine("local  x = " + x);
    }
}
```

[실행결과]

```
static x = 2
local  x = 4
```

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 [1/2]

- 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; // 큰 값
```

조건문 – if 문 [2/2]

□ 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;  
}
```

- 여기서, default의 의미는 otherwise
- break 문을 사용하여 탈출

반복문 – for 문 [1/3]

- 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 – for 문 [2/3]

□ for 문의 실행순서

반복문 – for 문 [3/3]

□ 무한 루프를 나타내는 for 문

```
for ( ; ; )  
    <문장>
```

- 루프 종료 : break 문, return 문

□ 내포된 for 문

- for 문 안에 for 문이 있을 때
- 다차원 배열을 다룰 때

```
for (i=0; i<N; ++i)  
    for (j=0; j<M; ++j)  
        matrix[i, j] = 0;
```

반복문 – while 문 [1/3]

□ while 문의 형태

```
while ( 조건식 )  
    <문장>
```

□ 예

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

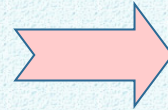
반복문 – while 문 [2/3]

□ while 문의 실행순서

반복문 – while 문 [3/3]

□ for 문과 while 문의 비교

```
for (i = 0; i < N; ++i)
    s += i;
```



```
i = 0;
while (i < N) {
    s += i;
    ++i;
}
```

- for --- 주어진 횟수
- while --- 주어진 조건

반복문 – do while 문

- 반복되는 문장을 먼저 실행 한 후에 조건식을 검사
- do-while 문의 형태

```
do
    <문장>
while ( 조건식 )
```

조건식이 거짓이라도 <문장>
부분이 적어도 한번은 실행

- precondition check --- for, while
- postcondition check --- do-while

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);
```

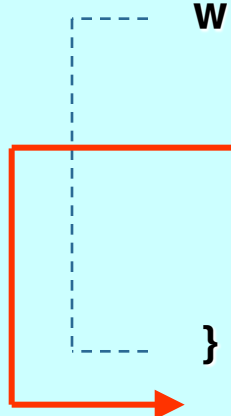

분기문 – break 문

- ❑ 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 [1/2]

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

```
continue;
```

- for 문 안에서 사용될 때

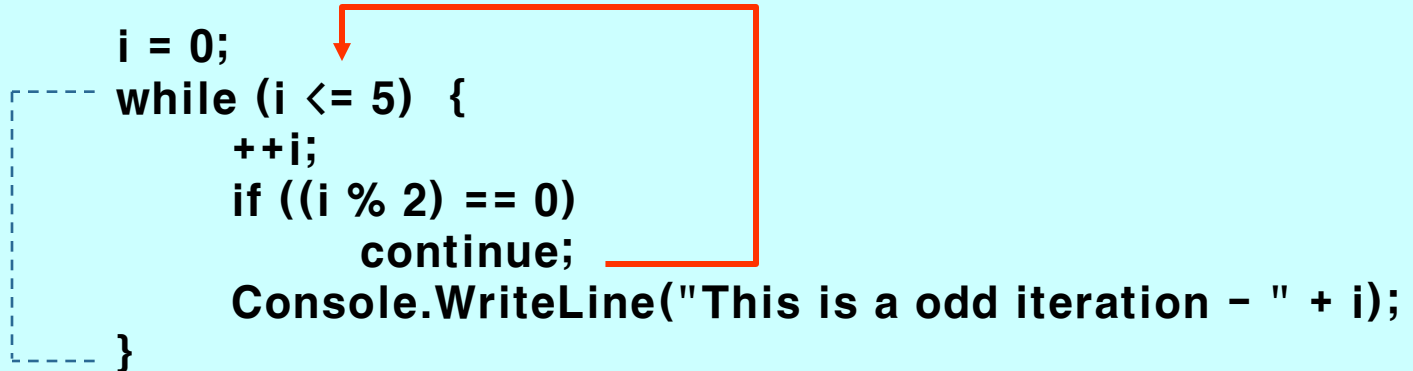
```
for (i=0; i<=5; ++i) {  
    if (i % 2 == 0)  
        continue;  
    Console.WriteLine("This is a " + i + " iteration");  
}
```



분기문 – continue 문 [2/2]

- while 문 안에서 사용될 때
 - 조건식을 검사하는 부분으로 옮김

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



[예제 3.16 – ContinueStApp.csc]

```
using System;
class ContinueStApp {
    public static void Main() {
        int i, max = int.MaxValue;
        try {
            Console.WriteLine("Start of try statement");
            i = max + 1;           // default: don't check Overflow.
            Console.WriteLine("After default overflow");
            unchecked {
                i = max + 1;      // don't check Overflow intensionally.
            }
            Console.WriteLine("After unchecked statement");
            checked {
                i = max + 1;      // check Overflow
            }
            Console.WriteLine("After checked statement");
        } catch (OverflowException e) {
            Console.WriteLine("caught an OverflowException");
        }
    }
}
```

[실행결과]

```
Start of try statement
After default overflow
After unchecked statement
caught an OverflowException
```

분기문 – goto 문

- 지정된 위치로 제어 흐름을 이동
- goto 문의 형태

```
goto label;  
goto case constant-expression;  
goto default;
```

- goto 문이 분기할 수 없는 경우
 - 외부에서 복합문 안으로 분기
 - 메소드 내에서 외부로 분기
 - finally 블록에서 블록 밖으로 분기

분기문 – return 문

- ❑ 메소드의 실행을 종료하고 호출한 메소드(caller)에게 제어를 넘겨주는 문장
- ❑ return 문의 형태

```
return;  
return <식>;
```

 [예제 3.17] 테스트

오버플로 검사문 – checked 문

- 오버플로를 명시적으로 검사하는 문장
 - System 네임스페이스에 있는 OverflowException 예외가 발생
- checked 문의 형태

```
checked {  
    // 오버플로가 발생하는지를 확인하려는 문장  
}
```

- 수식 checked 문의 형태

```
checked (오버플로가 발생하는지를 확인하려는 수식)
```

오버플로 검사문 – unchecked 문

- 오버플로를 의도적으로 검사하지 않을 경우
- unchecked 문의 형태

```
unchecked {  
    // 오버플로를 의도적으로 검사하지 않으려는 문장  
}
```

[예제 3.18 – OverflowApp.csc]

```
using System;
class OverflowApp {
    public static void Main() {
        int i, max = int.MaxValue;
        try {
            Console.WriteLine("Start of try statement");
            i = max + 1;           // default: don't check Overflow.
            Console.WriteLine("After default overflow");
            unchecked {
                i = max + 1;      // don't check Overflow intensionally.
            }
            Console.WriteLine("After unchecked statement");
            checked {
                i = max + 1;      // check Overflow
            }
            Console.WriteLine("After checked statement");
        } catch (OverflowException e) {
            Console.WriteLine("caught an OverflowException");
        }
    }
}
```

[실행결과]

```
Start of try statement
After default overflow
After unchecked statement
caught an OverflowException
```


표준 입출력 [1/4]

- 입출력 장치가 미리 정해진 입출력을 의미
- C# 언어의 기본 네임스페이스인 System으로부터 제공
- 표준 입력 메소드
 - Console.Read()
 - 키보드로부터 한 개의 문자를 읽어 그 문자의 코드값을 정수형으로 반환하는 기능
 - Console.ReadLine()
 - 한 라인을 읽어 string형으로 반환하는 기능
 - 숫자 값으로 바뀌어야 하는데 정수인 경우
 - int.Parse() 메소드 사용

[예제 3.20 – ReadLineApp.csc]

```
using System;
class ReadLineApp {
    public static void Main() {
        int time, hour, minute, second;
        Console.WriteLine("*** Enter an integral time : ");
        time = int.Parse(Console.ReadLine());
        hour = time / 10000;
        minute = time / 100 % 100;
        second = time % 100;
        Console.WriteLine("*** Time is " + hour + ":" + minute + ":" + second);
    }
}
```

[입력데이터]

```
*** Enter an integral time : 102030
```

[실행결과]

```
*** Time is 10:20:30
```

표준 입출력 [2/4]

□ 표준 출력 메소드

□ Console.Write()

- 화면에 매개 변수의 값을 출력

□ Console.WriteLine()

- 화면에 매개 변수의 값을 출력한 후 다음 라인으로 출력 위치를 이동

표준 입출력 [3/4]

- 형식화된 출력(formatted output)
 - 출력하려는 값에 포맷을 명시하여 원하는 형태로 출력
 - 출력 포맷의 형태

`{N[,W][:formatCharacter]}`

- N : 매개 변수를 위치적으로 지칭하는 정수 (단, 0부터 시작)
- W : 출력될 자릿수의 폭을 나타내며 선택으로 명시
 - '-' 기호를 붙이면 좌측정렬로 출력
- formatCharacter : 한 문자로 이루어진 형식 지정 문자를 의미

표준 입출력 [4/4]

- 형식 지정 스트링
 - 매개 변수의 개수와 일치하는 출력 포맷
- 표준 형식 지정문자

형식 지정자	설명
C 또는 c	통화 표시
D 또는 d	10진수 형태(정수형만 가능)
E 또는 e	지수 형태
F 또는 f	고정 소수점 형태
G 또는 g	고정 소수점 또는 지수 형태 중 간략한 형태를 선택한다.
N 또는 n	10진수(자릿수 구분을 위한 ‘,’ 포함)
P 또는 p	백분율(‘%’도 포함)
R 또는 r	결과 스트링을 다시 읽었을 때, 원 값과 동일함을 보장 (부동소수점 수만 가능)
X 또는 x	16진수(정수형만 가능)

[예제 3.22 – FormattedOutputApp.csc]

```
using System;
class FormattedOutputApp {
    public static void Main() {
        Console.WriteLine("1) {0,-5},{1,5},{2,5}\", 1.2, 1.2, 123.45);
        double d = Math.PI;
        Console.WriteLine("2) {0}\", d);
        Console.WriteLine("3) {0:C}\", d);
        Console.WriteLine("4) {0:E}\", d);
        Console.WriteLine("5) {0:F}\", d);
        Console.WriteLine("6) {0:G}\", d);
        Console.WriteLine("7) {0:P}\", d);
        Console.WriteLine("8) {0:R}\", d);
        Console.WriteLine("9) {0:X}\", 255);
    }
}
```

[실행결과]

- 1) 1.2 , 1.2, 123.45
- 2) 3.14159265358979
- 3) W3
- 4) 3.141593E+000
- 5) 3.14
- 6) 3.14159265358979
- 7) 314.16 %
- 8) 3.1415926535897931
- 9) FF