



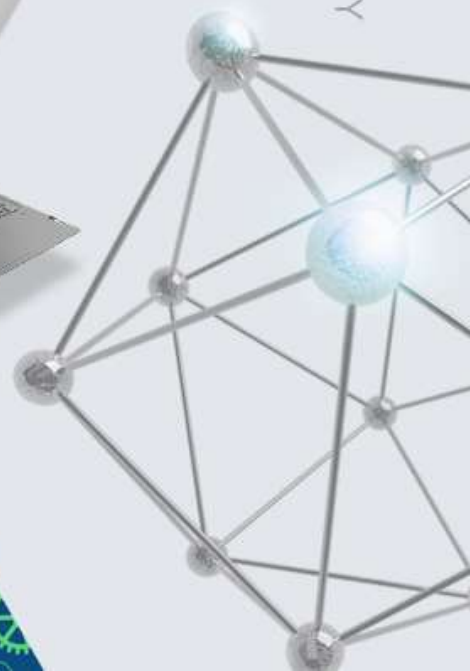
한국기술교육대학교
온라인평생교육원

The 4th Industrial Revolution is characterized by super connectivity and super intelligence, where various products and services are connected to the network, and artificial intelligence and information communication technologies are used in 3D printing, unmanned transportation, robotics. Of the world's most advanced technologies.

T
E
C
H
N
O
L
O
G
Y

C# 프로그래밍

문장



The 4th Industrial Revolution is characterized by super connectivity and super intelligence, where various products and services are connected to the network, and artificial intelligence and information communication technologies are used in 3D printing, unmanned transportation, robotics. Of the world's most advanced technologies.

문장



학/습/목/표

1. C# 언어의 문장의 종류를 이해하고 설명할 수 있다.
2. 배정문, 혼합문, 제어문을 효율적으로 구현할 수 있다.
3. 표준 입출력을 효율적으로 구현할 수 있다.



학/습/내/용

1. 문장의 종류
2. 배정문과 혼합문
3. 제어문
4. 오버플로우 검사문과 표준 입출력

1. 문장의 종류

1) 개요

배정문

혼합문

제어문

오버플로우
검사문

표준
입출력문

리소스문

동기화문

예외처리문

1. 문장의 종류

1) 개요

C# 언어의 문장

배정문

var = exp

혼합문

{ }

제어문

조건문: if문, switch문

반복문: for문, while문, do-while문, foreach문

분기문: break문, continue문, return문, goto문

오버플로우 검사문

checked문, unchecked문

표준 입출력문

Console.Read(), Console.ReadLine(),
Console.Write(), Console.WriteLine()

리소스문

using문

동기화문

lock문

예외처리문

try-catch-finally문

2. 배정문과 혼합문

1) 배정문

(1) 의미

- 배정문: 값을 변수에 저장하는데 사용

(2) 형태

- `<변수> = <식>;`
→ 형태는 C 언어와 같음
- `a = b = exp;`
→ 좌측 부분이 여러 번 나오는 경우 exp가 먼저 연산되어,
그 결과가 b에 할당되고 다시 b값이 a에 할당됨
- 변수의 초기값을 주는 데 유용함

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

2. 배정문과 혼합문

1) 배정문

(3) 형 변환

- 묵시적 형 변환 : 컴파일러에 의해 자동 변환됨
- 명시적 형 변환 : 프로그래머가 캐스트(Cast) 연산자를 사용하여 변형해야 함

```
namespace AssingmentStApp
{
    class Program
    {
        static void Main(string[] args)
        {
            short s;
            int i;
            float f;
            double d;

            s = 526;
            d = f = i = s;
            Console.WriteLine("s = " + s + " i = " + i);
            Console.WriteLine("f = " + f + " d = " + d);
        }
    }
}
```

2. 배정문과 혼합문

2) 혼합문

(1) 의미

- 혼합문 : 여러 문장을 한데 묶어 하나의 문장으로 나타냄
→ 주로 문장의 범위를 표시함

(2) 형태

- { <선언> 또는 <문장> }

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

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

- 예제 : LocalVariableApp.cs

```
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);
    }
}
```

실행 결과

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

2. 배정문과 혼합문

2) 혼합문

(3) 지역변수(Local Variable)

- 지역변수의 특징
 - 블록의 내부에서 선언된 변수
 - 선언된 블록 안에서만 참조 가능

3. 제어문

1) 개요

(1) 제어문

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

3. 제어문

2) 조건문

(1) 조건문 의미

- 주어진 조건에 따라 수행되는 부분이 다를 때 사용하는 문장

(2) if 문

- if 문 형태

```
if ( <조건식> ) <문장>
```

```
if ( <조건식> ) <문장1> else <문장2>
```

→ 조건식의 연산결과 : 논리형 (true or false)

- 예시

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

3. 제어문

2) 조건문

(2) if 문

- 참 부분에서 if 문이 재등장

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

- else 부분에서 if 문이 재등장

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

3. 제어문

2) 조건문

(3) switch 문

▪ if 문

조건식의 결과에 따라
참일 때 실행하는 부분과
거짓일 때 실행할 부분

VS

▪ switch 문

조건에 따라 여러 경우로
나누어 각각을
처리해야 되는 경우

▪ switch 문의 형태

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

→ default의 의미는 otherwise

→ 반드시, break 문을 사용하여 탈출
(단, C/C++에서는 붙이지 않아도 됨)

3. 제어문

3) 반복문

(1) 반복문 의미

- 프로그램의 일정한 부분을 주어진 조건이 만족될 때까지 반복해서 실행하는 문장
- 반복문의 종류 : for 문, while 문, do-while 문, foreach 문

(2) for 문

- 정해진 횟수만큼 일련의 문장을 반복 실행
→ 일반적으로 가장 널리 사용되는 반복문
- For 문의 형태

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

→ <식1> : 제어 변수 초기화

→ <식2> : 제어 변수를 검사하는 조건식

→ <식3> : 제어 변수의 값을 수정

→ <문장> : 루프 몸체, <식 2>의 조건식이 참인 동안 반복 실행되는 부분

예

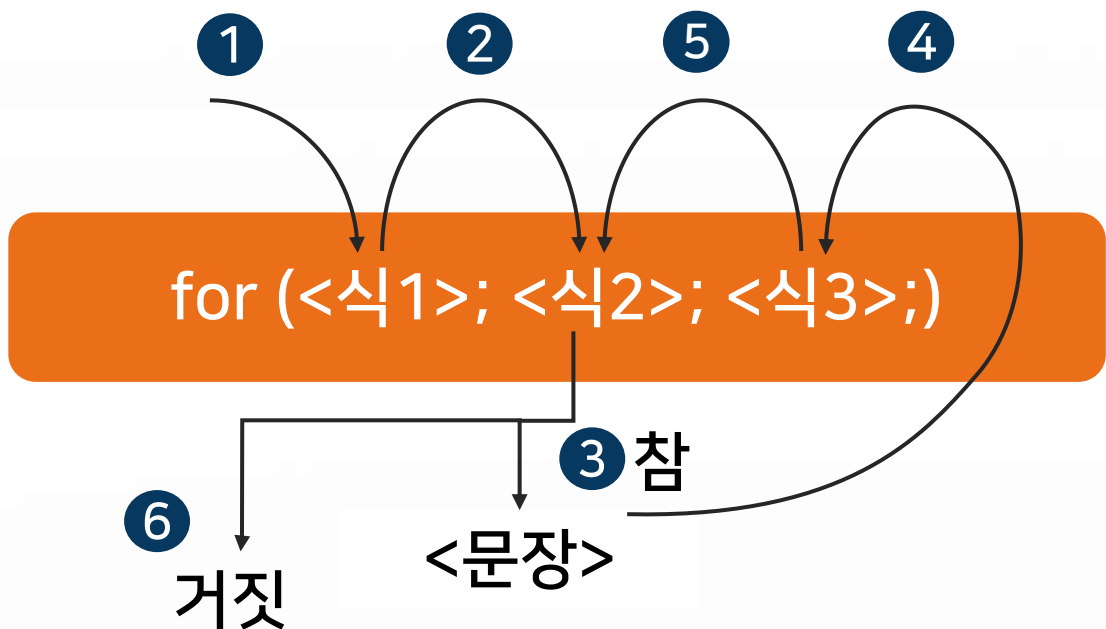
```
s = 0;
for (i = 1; i <= N; ++i)
    // 1부터 N까지의 합 : i 증가
    s += i;
```

3. 제어문

3) 반복문

(2) for 문

- for 문의 실행 순서



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

3. 제어문

3) 반복문

(2) for 문

- 무한 루프를 나타내는 for 문

```
for ( ; ; )  
    <문장>
```

→ 루프 종료 : break문, return문

- 내포된 for 문

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

→ for 문 안에 for 문이 있을 때, 다차원 배열을 다룰 때

3. 제어문

3) 반복문

(3) while 문

- 주어진 조건식이 참인 경우에만 프로그램의 일정한 부분을 반복해서 실행하는 반복문
- While 문의 형태

```
while ( 조건식 )  
    <문장>
```

예

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

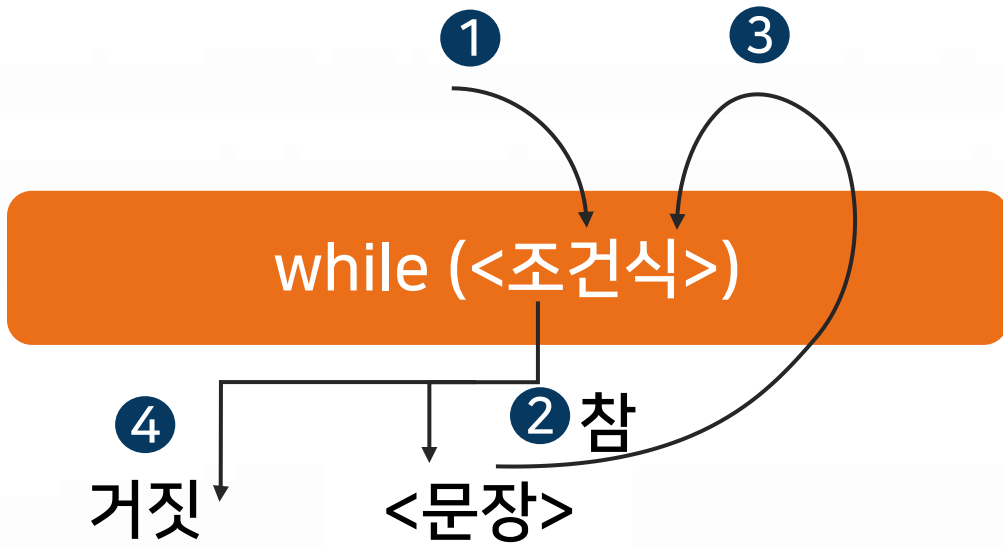
반복 몸체가 2개 이상의 문장이면 그 문장들이 while 문에 속해 있다는 것을 나타내기 위해 블록인 {}로 표현

3. 제어문

3) 반복문

(3) while 문

- While 문의 실행 순서



- for 문과 while 문의 비교

For 문

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

주어진 횟수

While 문

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

주어진 조건

3. 제어문

3) 반복문

(4) do-while 문

- 반복되는 문장을 먼저 한 번 실행 한 후에 조건식을 검사하여 참이면 계속 반복, 거짓이면 반복 종료
- do-While 문의 형태

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

조건식이 거짓이라도 <문장>
부분이 적어도 한번은 실행

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

3. 제어문

3) 반복문

(5) foreach 문

- 조건식을 검사하는 반복문과는 달리 데이터의 집합에 대한 반복을 수행
→ 데이터의 집합을 구성하는 원소를 순차적으로 처리할 때 유용
- foreach 문의 형태

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

예

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

3. 제어문

4) 분기문

(1) 분기문 의미

- 지정된 곳으로 제어를 옮기는 역할
- 분기문의 종류 : break 문, goto 문, continue 문, return 문

(2) break 문

- 반복문에서 현재의 실행을 멈추고 자기를 감싸고 있는 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;
}
```

3. 제어문

4) 분기문

(3) continue 문

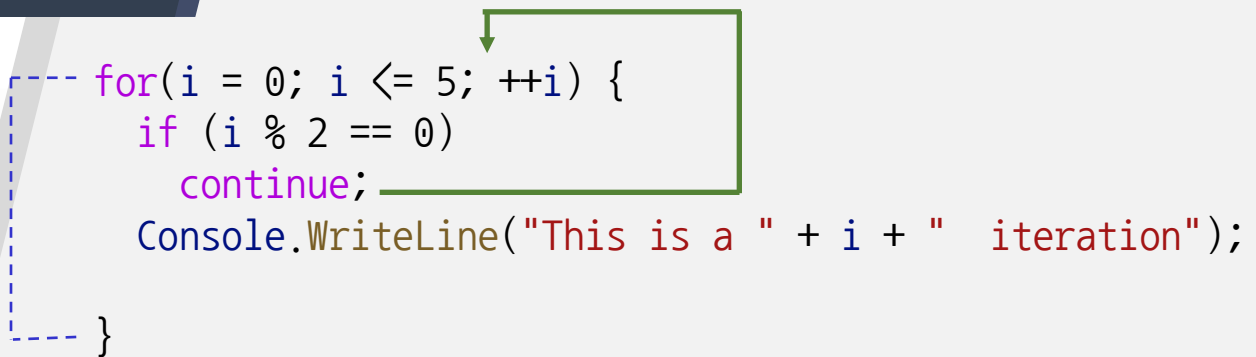
- 다음 반복이 시작되는 곳으로 제어를 옮기는 기능
- continue 문의 형태

```
continue;
```

- for 문 안에서 사용될 때

예

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



3. 제어문

4) 분기문

(4) goto 문

- 지정된 위치로 제어 흐름을 이동하는 문장
- goto 문의 형태

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

레이블(label) : 지정된 위치를 나타내기 위한
방법으로 명칭 형태

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

3. 제어문

4) 분기문

(5) return 문

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

`return;` ————— 제어만 넘겨줌

`return <식>;` ————— 반환값과 함께 제어를 넘겨줌

4. 오버플로우 검사문과 표준 입출력

1) 오버플로우 검사문

(1) 오버플로우

- 정수식 연산의 결과가 정수형이 표현할 수 있는 범위를 넘어가는 것
- 잘못된 계산 결과를 초래할 수 있기 때문에 정수식에서 오버플로우가 일어나는 지를 명시적으로 검사하여 처리하는 것이 바람직함

(2) checked 문

- 오버플로우를 명시적으로 검사하는 문장
→ System 네임스페이스에 있는 `OverflowException` 예외가 발생

- checked 문의 형태

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

- 수식 checked 문의 형태

```
checked (오버플로우가 발생하는지를 확인하려는 수식)
```


4. 오버플로우 검사문과 표준 입출력

1) 오버플로우 검사문

(3) unchecked 문

- 오버플로우를 의도적으로 검사하지 않을 경우
- unchecked 문의 형태

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

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(1) 입출력문

1

입출력

입력과 출력

2

입력

외부 입력 장치로부터
데이터를 프로그램으로
읽어 들이는 행위

3

출력

프로그램상의 어떤
값을 출력 장치로
내보내는 행위

■ 입출력문의 의미

→ 입출력 장치가 미리 정해진 입출력을 의미

→ C# 언어의 기본 네임스페이스인 system으로부터 제공

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(1) 입출력문

▪ 표준 입력 메소드

- `Console.Read()` : 키보드로부터 한 개의 문자를 읽어 그 문자의 코드값을 정수형으로 반환하는 기능
- `Console.ReadLine()` : 한 라인을 읽어 string형으로 반환하는 기능, 숫자값으로 바꿔야하는데 정수인 경우 `int.Parse()` 메소드 사용

▪ 표준 출력 메소드

- `Console.Write()` : 화면에 매개 변수의 값을 출력
- `Console.WriteLine()` : 화면에 매개 변수의 값을 출력한 후 다음 라인으로 출력 위치를 이동

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(1) 입출력문

▪ 예제 : ReadLineApp.cs

```
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);
    }
}
```

입력 데이터

*** Enter an integral time : 102030

실행 결과

*** Time is 10:20:30

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(2) 형식화된 출력

- 출력하려는 값에 형식을 명시하여 원하는 형태로 출력하는 것
- 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

- 형식 지정 스트링 : 매개 변수의 개수와 일치하는 출력 포맷

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(2) 형식화된 출력

▪ 표준 형식 지정 문자

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 선택함
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)

4. 오버플로우 검사문과 표준 입출력

2) 표준 입출력

(2) 형식화된 출력

▪ 예제 : FormattedOutputApp.cs

```
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);
    }
}
```



1. 문장의 종류

- 배정문, 혼합문, 제어문, 오버플로우 검사문, 표준 입출력문, 리소스문, 동기화문, 예외처리문

2. 배정문과 혼합문

- 프로그래밍 언어에서 배정문은 가장 기본적인 문장으로 새롭게 계산되어진 값을 변수에 저장하는데 사용됨
- 혼합문은 여러 문장을 한데 묶어 하나의 문장으로 나타내는 역할을 담당함



3. 제어문

- 제어문은 프로그램의 실행 순서를 바꾸는 데 사용하며 실행 순서를 제어하는 방법에 따라 조건문, 반복문 그리고 분기문으로 나뉨
- 조건문 : 주어진 조건에 따라 수행되는 내용이 다를 때 사용하는 문장임 (if, switch)
- 반복문 : 정해진 횟수만큼 또는 주어진 조건이 만족하는 동안 반복해서 실행하는 문장임 (for, while, do-while, foreach)
- 분기문 : 제어를 지정된 곳으로 옮기는 문장임 (break, continue, return, goto)

4. 오버플로우 검사문과 표준 입출력

- 오버플로우 : 정수식 연산의 결과가 정수형이 표현할 수 있는 범위를 넘어가는 것
- 오버플로우 검사문 : 오버플로우를 명시적으로 검사하는 문장임
- 입출력문 : 입출력 장치가 미리 정해진 입출력을 의미하며, C# 언어의 기본 네임스페이스인 System으로부터 제공함
- 표준 입출력 메소드 : `Console.Read()`, `Console.ReadLine()`, `Console.Write()`, `Console.WriteLine()`