



## [7주차 2강] 반복문(4)



# 학습 내용

## 6.6 반복문 기타



# 학습 목표

6.6 break 문, continue 문을 이용한 흐름 제어를 익힌다.





## 6.6 반복문 기타



반복문의 실행 상태를 직접 제어하고자 break 문과 continue 문을 사용한다.

### break 문

- 현재 사용 중인 반복문을 중단하고 제어를 반복문 바깥으로 이동

```
for (cnt = 1; cnt < 10; cnt++)
```

```
    if (cnt % 4 == 0 )
```

```
        break;
```

```
    printf("cnt 값은 %d 입니다\n", cnt);
```

cnt 값은 4 입니다

- ✓ cnt 값이 1 에서 부터 증가하다, 4가 됐을 때,  
if 문이 참이 되고 break 문 수행(for 문을 빠져나옴)



### break 문 동작 과정

```
while( 조건식 ) {
```

```
...
```

```
break;
```

```
...
```

```
}
```

```
문장 1;
```

```
do {
```

```
...
```

```
break;
```

```
...
```

```
} while( 조건식 );
```

```
문장 1;
```

```
for( 초기식; 조건식 ; 증감식 ) {
```

```
...
```

```
break;
```

```
...
```

```
}
```

```
문장 1;
```



### continue 문

- 현재 수행 중인 반복문에서 현재 조건 값에 대한 처리를 중단하고, 다음 조건 값에 대한 처리를 수행  
✓ 결과적으로 continue 문과 반복문의 마지막 부분 사이에 있는 문장은 실행되지 않음

```
cnt = 0;
for ( i = 1; i < 10; i++) {
    if (i % 4 != 0 )
        continue;
    cnt++;
}
```

4의 배수는 2 개 입니다

```
printf("4의 배수는 %d 개 입니다\n", cnt);
```

- ✓ i 값이 4, 8 일 때 continue 문 수행(cnt++ 문장을 건너뛰)
- ✓ 4의 배수의 개수를 구하는 바람직한 코드는 아님



### continue 문 동작 과정

```
while( 조건식 ) {
```

```
...
```

```
    continue;
```

```
...
```

```
}
```

```
문장 1;
```

```
do {
```

```
...
```

```
    continue;
```

```
...
```

```
} while( 조건식 );
```

```
문장 1;
```

```
for( 초기식; 조건식 ; 증감식 ) {
```

```
...
```

```
    continue;
```

```
...
```

```
}
```

```
문장 1;
```

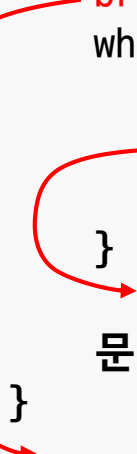




### 중첩 반복에서 break 문과 continue 문

- 그 문장을 둘러싸고 있는 가장 안쪽 반복문에 대해 적용됨

```
for( ... ) {  
    ...  
    break; ⇒ for 문 종료, 이후 문장2 수행  
    while( ... ) {  
        ...  
        break; ⇒ while 문 종료, 이후 문장1 수행  
        ...  
    }  
    문장 1;  
}  
문장 2;
```





### 무한 반복

- 일반적으로 반복문에서는 조건을 지정하여 조건에 맞는 경우에만 반복을 시킴
- 경우에 따라서는 반복이 무한히 지속되는 **무한 반복**을 사용하는 경우도 있음
  - ✓ 조건식을 항상 참이 되도록 설정
  - ✓ 보통 다음 형태 사용

```
while( 1 ) {  
    반복할 문장;  
}
```

```
for( ; ; ) {  
    반복할 문장;  
}
```

### 무한 반복을 이용한 반복문 예

```
/* 일반적인 반복 형태 */  
while( i < 10 ) {  
    printf("%d\n", i);  
    i++;  
}
```

```
/* 무한 반복을 이용한 형태 */  
while ( 1 ) {  
    if( i >= 10 ) break;  
    printf("%d\n", i);  
    i++;  
}
```

### break 문, continue 문, 무한 반복

- 반복문 중간에 제어를 마음대로 조정할 수 있어 편리
- But, 남용할 경우 프로그램의 가독성에 악영향
  - ✓ 위 두 코드 중 어느 것이 코드의 동작을 이해하기 편한가?



### 반복문 작성 시 "강력" 권장사항

- break 문, continue 문, 무한 반복을 사용하지 **않고** 코딩하라.
- 반복문의 형식에 맞추어 코드를 작성하라.

✓ 편한대로 코드를 작성하다 보면,

반복 조건이나 반복이 종료된 후에 할 일을

반복문 내부에 작성하는 경우 많음 → 바람직하지 않은 코드

```
/* 반복문의 논리 구조 */
```

```
while( 반복 조건 ) {
```

```
    여기에는 반복할 일만...
```

```
}
```

```
반복이 종료된 후 할 일은 여기에...
```



### 나쁜 코드와 좋은 코드 예시

- 문제: 입력된 정수들의 합을 구한 후, 합이 양수인지 홀수인지 출력하시오.
  - ✓ 정수는 5개가 입력되거나 또는 0이 입력될 때까지 입력 받는다.

입력 예시 1

4 5 -2 4 7

출력 예시 1

18  
even

입력 예시 2

4 5 0

출력 예시 2

9  
odd



### 나쁜 코드 예시

```
int x, sum = 0;
int i = 0;

while( i < 5 ) {
    scanf("%d", &x);
    sum += x;
    i++;

    if( i == 5 || x == 0 ) {
        printf("%d\n", sum);
        if( sum % 2 )
            printf("odd\n");
        else
            printf("even\n");
        break;
    }
}
```



### 좋은 코드 예시

```
int x = -1, sum = 0;
int i = 0;

while( i < 5 && x != 0 ) {
    scanf("%d", &x);
    sum += x;
    i++;
}

printf("%d\n", sum);
if( sum % 2 )
    printf("odd\n");
else
    printf("even\n");
```



### 코딩을 잘하려면?

- **가독성** 좋은 코드를 작성해야 함
- Why?
  - ✓ 논리 구조가 간단해 저서 이해하기 쉬움
  - ✓ 코드가 단순해짐
  - ✓ 버그 생길 가능성도 적고, 디버깅하기도 쉽고
  - 결국, 프로그램을 정확하고 빠르게 작성할 수 있음

**"세 살 코딩 버릇 여든까지 간다"**

처음부터 코드를 효율적이고 간결하게 작성하는 습관을 들이자!!!

# 학습 정리

- 반복문에서는 **break 문**과 **continue 문**을 이용하여 반복문의 실행을 직접 제어할 수 있음
- **break 문**은 현재 수행 중인 반복문을 중단하고 프로그램 제어를 반복문 다음으로 이동시킴
- **continue 문**은 현재 조건 값에 대한 처리만 중단하고, 제어를 반복문의 마지막으로 이동하고 다시 반복문 처음으로 돌아가 이후 반복을 계속 수행함
- **중첩 반복**에서 break 문과 continue 문 그 문장을 둘러싸고 있는 가장 안쪽 반복문에 대해 적용됨