

반복문

소프트웨어학부
박영훈 교수

이 단원의 목표

- 반복문

`while`, `do - while`, `for`문

- `break`

- `continue`

- `goto`

- Null statement

C 프로그래밍에서 반복문

- 반복문이란, 특정한 몇 개의 명령문을 반복해서 수행하게 하는 것을 말한다.
- C 프로그래밍에서는 모든 반복문에 반복 조건이 있다.
- 즉, 반복 조건이 참일 때 반복문 안에 있는 명령문들을 수행하고, 거짓이면 반복문을 빠져 나온다.
- C 프로그래밍에서 제공하는 반복문들

`while`

`? do-while`

`for`

while 문

- C 프로그래밍에서 가장 기본적인 반복문
- **while** 문은 다음과 같은 형태로 사용된다.

```
while ([수식]) [명령문];
```

혹은

수식이 참이면 (0이 아니면) 실행
거짓이면 빠져나감.

```
while ([수식]) {  
    [명령문 1];  
    [명령문 2];  
    ...  
}
```

- **while** 문 안의 수식을 controlling expression 이라 하고, **while** 문에 포함되어 있는 명령문을 loop-body라 한다.

while 문

- `while` 문의 예:

```
while (i < n)           // controlling expression
    i = i * 2;         // loop-body
```

- `while` 문은 다음 순서로 실행된다.

1. controlling expression을 계산
2. 계산 결과가 0이 아니면 loop-body를 실행. 0이면 `while`을 빠져나옴.
3. Step 1로 돌아감

- 만일 controlling expression이 맨 처음부터 0이었다면 loop-body는 실행되지 않는다.
- 만일 controlling expression이 항상 0이 아니라면 무한 루프를 돌게 된다.

예:

```
while (1) ...
```

프로그래밍이 안됨.

while 문

- 다음과 같은 **while** 문이 주어져 있다고 하자.

!g

:과거에 썼던 for문과
실용성

✓

```
int i = 1, n = 10;
while (i < n)      // controlling expression
    i = i * 2;     // loop-body
```

- 그러면, 다음 순서로 실행된다.

i < 10 인지 검사	Yes
i에 i * 2의 값을 대입	i는 2
i < 10 인지 검사	Yes
i에 i * 2의 값을 대입	i는 4
i < 10 인지 검사	Yes
i에 i * 2의 값을 대입	i는 8
i < 10 인지 검사	Yes
i에 i * 2의 값을 대입	i는 16
i < 10 인지 검사	No
while문을 빠져나옴	

while 문

- **while** 문에서 loop-body는 복합명령문으로 구성될 수도 있다. 이 때는 중괄호로 묶어 준다.

- 예:

```
int s = 0, i = 10;
while(i > 0){
    s += i;
    i--;
}
```

- 위 예제에서 **while** 문이 끝나면 s에는 55가 저장된다. *(0 ~ 1 더해주니까)*

while 문 실습 1

- 자연수를 입력받아, 1부터 그 자연수까지 그 수와 제곱수를 나란히 표시
- 실시 예:

Enter number of entries in table: 5

1	1
2	4
3	9
4	16
5	25

```
#include <stdio.h>
int main(void){
    int n, i = 1;
    printf("Enter number of entries in table: ");
    scanf("%d", &n);
    while(i <= n){
        printf("%d\t%d\n", i, i*i);
        i++;
    }
    return 0;
}
```

```
#include <stdio.h>
int main(void){
    int n, i = 0;
    printf("Enter number of entries in table: ");
    scanf("%d", &n);
    while(++i <= n) printf("%d\t%d\n", i, i*i);
    return 0;
}
```

```
#include <stdio.h>
int main(void){
    int n, i = 0;
    printf("Enter number of entries in table: ");
    scanf("%d", &n);
    while(i++ < n){
        printf("%d\t%d\n", i, i*i);
    }
    return 0;
}
```

✗
++i로 하면
처음부터 실행될지 모름.

while 문 실습 2

- 여러 개의 정수를 키보드로 입력 받아 그 정수들의 합을 표시하는 프로그램을 작성
- 각 정수는 white space로 구분
- 정수 0이 입력되면 그 전까지 입력된 정수들의 합을 출력
- 실시 예:

Enter integers (0 to terminate): 8 23 71 5 0
The sum is 107.

```
#include <stdio.h>
int main(void){
    int x = 1, s = 0;
    printf("Enter integers (0 to terminate): ");
    while(x != 0){
        scanf("%d", &x);
        s += x;
    }
    printf("The sum is %d\n", s);
    return 0;
}
```

scanf("%d", &x);

do - while 문

while문은 아예 실행이 안될 수도 있고
적어도 한번은 꼭 실행될

- **do - while** 문은 다음과 같은 형태로 사용된다.

```
do [명령문]; while ([수식]);
```

혹은

```
do{  
    [명령문 1];  
    [명령문 2];  
    ...  
} while ([수식]);
```

- **do - while** 문에서 명령문들을 loop-body, 수식을 controlling expression이라 한다.
- **do - while** 문에서는 우선 loop-body가 먼저 실행되고, 그 다음에 controlling expression이 실행된다.
- controlling expression의 결과값이 0이면 **do - while** 문을 빠져나온다.
- controlling expression의 결과값이 0이 아니면 위의 작업들을 다시 수행한다.

while 문과 do - while 문의 관계

- 다음 while 문

변형할때 주의하기

```
int s = 0, i = 10;
while(i > 0){
    s += i;
    i--;
}
```

은 다음과 같이 do - while 문으로 바꿀 수 있다.

```
int s = 0, i = 10;
do{
    s += i;
    i--;
}while(i > 0);
```

- while 문과의 다른 점은, do - while 문에서는 loop-body가 최소한 최초 한 번은 실행 된다는 것이다.
- do - while 문에서 마지막 while 다음에는 반드시 세미콜론 (;)이 있어야 한다.

do - while 문 실습

- 음이 아닌 정수를 키보드로 입력받아, 그 정수의 자릿수를 출력한다.
- 0도 한 자리 정수로 간주한다.
- 실시에

Enter a non-negative integer: 60
The number has 2 digit(s).

```
#include <stdio.h>
int main(void){
    int n, d = 0;
    printf("Enter a non-negative integer: ");
    scanf("%d", &n);
    do{
        n /= 10;
        d++;
    } while(n > 0);
    printf("The number has %d digit(s)\n", d);
    return 0;
}
```

for 문

- C 프로그래밍에서 가장 많이 쓰이는 반복문.
- 기본적인 형태는 다음과 같다.

```
for ([수식 1]; [수식 2]; [수식 3]) [명령문];
```

↓
초기값 조건 업데이트

또는

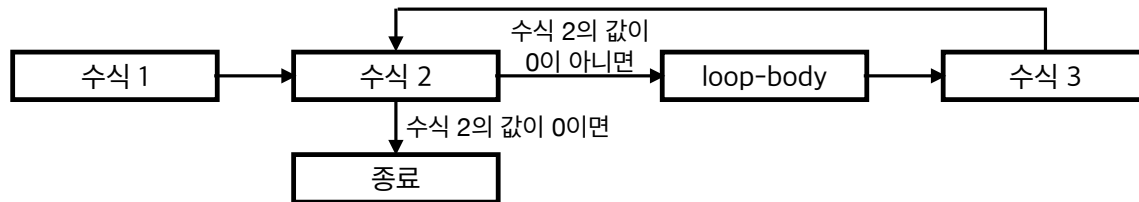
```
for ([수식 1]; [수식 2]; [수식 3]) {  
    [명령문 1];  
    [명령문 2];  
    ...  
}
```

- [수식 1]은 최초 실행 단계, [수식 2]는 반복문 조건이며, [수식 3]은 모든 loop-body가 실행되고 난 후에 실행되는 문장이다.

for 문의 실행 순서

- 다음 **for** 문에서 실행 순서는 다음과 같다.

```
for([수식 1]; [수식 2]; [수식 3]){  
    [명령문 1];  
    [명령문 2];  
    ...  
}
```



- 위 **for** 문을 **while** 문으로 바꾸면 다음과 같다.


```
[수식 1];  
while([수식 2]){  
    [명령문 1];  
    [명령문 2];  
    ...  
    [수식 3];  
}
```

for 문의 사용법

- 다음 `while` 문

```
int s = 0, i = 10;
while(i > 0){
    s += i;
    i--;
}
```

을 다음과 같은 `for`문으로 바꿀 수 있다.



```
int s, i;
for(i = 10, s = 0 ; i > 0; i--)
    s += i;
```

- `for` 문에서 [수식 1]과 [수식 3]에 여러 개의 수식이 들어갈 수 있다. 이 때, 각 수식은 쉼표(,)로 구분해준다.

for 문의 사용 예

- **for**문은 정수가 1씩 증가 또는 감소하는 순서대로 실행해야 할 때 매우 유용하다.
- 각각의 경우에 대한 **for**문의 사용 예는 다음과 같다.
- **i가 0부터 n-1일 때까지 반복:** `for(i = 0 ; i < n ; i++){ ...`
- **i가 1부터 n일 때까지 반복:** `for(i = 1 ; i <= n ; i++){ ...`
- **i가 n-1부터 0일 때까지 반복:** `for(i = n - 1 ; i >= 0 ; i--){ ...`
- **i가 n부터 1일 때까지 반복:** `for(i = n ; i > 0 ; i--){ ...`
- 흔히 하기 쉬운 실수들
 - i가 증가하는데 조건식이 `i > [변수]` 이거나 `i >= [변수]` 로 되어 있는 경우
 - i가 감소하는데 조건식이 `i < [변수]` 이거나 `i <= [변수]` 로 되어 있는 경우
 - `<=`, `>=`를 써야 할 때 각각 `<`, `>`를 쓰는 경우, 혹은 그 반대의 경우

C/C++ : 리눅스에서 강제할 필요
↳ 맥하드웨어에서 나오는 버그

for 문에서 수식을 빼고 쓰는 기법

- **for** 문에 포함되는 세 가지 수식 중에서 필요가 없는 경우는 비워놔도 된다.
- **for**(; [수식 2] ; [수식 3])
초기화 수식이 없는 경우. 초기화 단계가 없다.
- **for**([수식 1] ; ; [수식 3])
이 경우는 반복 조건이 없는 경우로, 무조건 참으로 간주한다.
따라서 **for**문 안에 **for**문을 빠져나가게 할 수 있는 다른 명령어(**break**, **return**, **goto** 등)들이 필요하다.
- **for**([수식 1] ; [수식 2] ;)
마지막 수식이 없는 경우.
- 아무 수식이 필요 없는 경우는 모두 비워도 무방하다. 즉, **for**(;;) 으로 두면 되며, 이 경우 무한루프를 돌게 된다.

쉼표 (,) 연산자

- **for**문에서 [수식 1], [수식 2], [수식 3]에는 각각 둘 이상의 수식이 들어갈 수 있다.

- 이 때는 여러 수식들을 쉼표로 구분해준다. 예를 들어,

```
for([수식 1-1], [수식 1-2], ... ; [수식 2-1], [수식 2-2], ... ;  
    [수식 3-1], [수식 3-2], ...) { ...
```

- 쉼표 연산자는 쉼표로 구분된 모든 수식이 계산되며, 계산 순서는 왼쪽부터이다.

- 쉼표 연산자도 부가기능이 있는데, 쉼표 연산자가 있는 수식들이 가지는 값은 마지막 수식의 결과값이다. 예를 들어,

```
i = 1;  
k = (j = 3, i++, i + j);
```

라는 수식에서 **j**에 3이 우선 대입되고, **i**가 1 증가하며, **i** + **j**의 값은 5가 된다. 이 때, **k**에는 마지막 수식 **i** + **j**의 결과값인 5가 저장되게 된다.

- 쉼표 연산자를 잘 활용하면 소스코드를 간단하게 만들 수 있다.

break 문

- 반복문 (**while**, **do - while**, **for**) 이나 **switch**문을 빠져나가는 명령

- 사용 예:

```
for(d = 2 ; d < n ; d++){  
    if(n % d == 0){  
        break;  
    }  
}  
printf("%d\n", d);
```

다가의약면

```
#include <stdio.h>  
int main(void){  
    int n;  
    while(1){  
        printf("Enter a positive integer: ");  
        scanf("%d", &n);  
        if(n > 0) break;  
        printf("Error!\n");  
    }  
    printf("n = %d\n", n);  
    return 0;  
}
```

for(;;)와 비슷

- 반복문 혹은 **switch**문이 여러 겹이면 하나만 빠져나간다. 예를 들어,

```
while(...){  
    switch(...){  
        ...  
        break;  
    }  
    ...  
}
```

- 에서 **break**;문에 의해서는 **switch**만 빠져나가고, **while** 안에는 그대로 남게 된다.

continue 문

- 반복문 안에서 loop-body의 맨 끝으로 점프시키는 명령문
- break문과는 달리 switch문에서는 못 쓴다.

- 사용 예:

```
int i, n = 0, sum = 0;
while(n < 10){
    scanf("%d", &i);
    if(i == 0) continue;
    sum += i;
    n++;
    /* continue jumps to here */
}
```

```
for(i = 1, k = 0 ; i <= n ; i++){
    if(n % i != 0) continue;
    k++;
    /* continue jumps to here */
}
```

goto 문

아래만 들어야함

- 강제로 미리 표시된 라벨로 점프시키는 명령문.
- 라벨 지정 방법 - 라벨 이름 뒤에 콜론(:)을 쓴다.

[라벨 이름]:

- 그런 후에 `goto [라벨 이름];` 이라 명령을 내리면 그 라벨로 점프한다.
- 사용 예:

```
for(d = 2 ; d < n ; d++){  
    if(n % d == 0){  
        goto done;  
    }  
}  
done:  
printf("%d\n", d);
```

goto 문

- 반복문을 if문과 goto문의 조합만으로도 만들 수 있다. 예를 들어

```
int s = 0, i = 10;
while (i > 0) {
    s += i;
    i--;
}
```

또는

```
int s = 0, i = 10;
L0:
if (i <= 0) goto L1;
s += i;
i--;
goto L0;
L1:
```

```
d = 0;
do {
    n /= 10;
    d++;
} while (n > 0);
```

```
d = 0;
L0:
n /= 10;
d++;
if (n > 0) goto L0;
```

- goto 문이 유용한 경우: 반복문 여러 겹을 한꺼번에 빠져나갈 때.
- 하지만, goto문을 많이 쓰면 소스코드의 가독성이 심각하게 떨어지게 되므로 되도록이면 쓰지 않는 것이 좋다.

무한루프의 위험도...??? ㅋㅋ.. 브레이크도..

null statement

- 세미콜론 (;) 으로만 구성된 명령문을 null statement라 한다.
- null statement를 잘 활용하면 소스코드를 간단하게 줄일 수 있다. 예를 들어,

```
for(i = 2 ; i <= n ; i++){  
    if(n % i == 0) break;  
}
```

는 다음과 같이 간단히 할 수 있다.

```
for(i = 2 ; i <= n && n % i != 0 ; i++);
```

