

반복문과 선택문

제주대학교 컴퓨터공학과

변영철 교수

(ycb@jejunu.ac.kr)

이 장을 공부하면

- 반복문과 선택문을 이해할 수 있다.
- C 프로그램을 작성 시 반복문과 선택문을 활용하여 응용 프로그램을 작성할 수 있다.

while 문

```
while (a < 10) ← 조건식
{
    a = a * 2; ← 실행문
}
```

- 조건식 먼저 검사
- 참이면 아래 실행문 실행
- 다시 위로 올라가서 **반복**

while 문



```
int main(void)
{
    int a = 1;

    while (a < 10)
    {
        a = a * 2;
    }
    printf("a : %d\n", a);

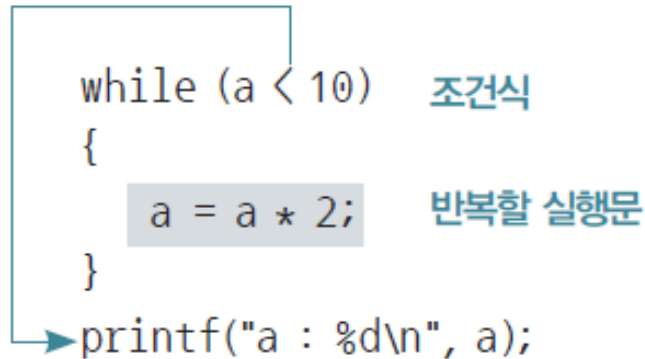
    return 0;
}
```

while 문

조건식이 참이 아니면 while문 끝나고
아래 문장 실행

조건식이 거짓이면
반복문 이후의
문장으로
건너뛴다.

```
while (a < 10)   조건식
{
    a = a * 2;   반복할 실행문
}
printf("a : %d\n", a);
```

A flowchart illustrating the execution of a while loop. It starts with the condition 'while (a < 10)' labeled '조건식' (Condition). If the condition is true, it enters a block containing 'a = a * 2;' labeled '반복할 실행문' (Statement to be repeated). After the block, it reaches 'printf("a : %d\n", a);'. A blue arrow loops back from the 'printf' statement to the 'while' condition, indicating the loop's continuation. If the condition is false, the flow proceeds straight down to the 'printf' statement.

while 문

들여쓰기 하면 코드를 이해하기 쉬움 (가독성)

```
while (a < 10)
{
    들여쓰기 → a = a * 2;
}
```

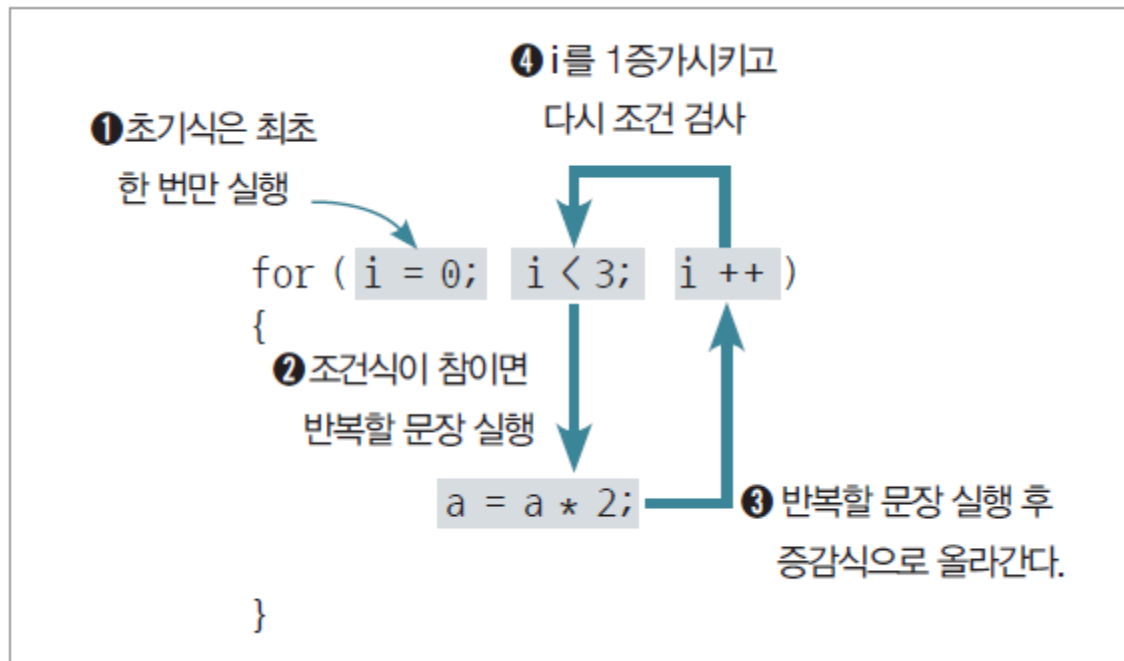
for 문

아래와 같이 초기식, 조건식, 증감식으로 반복

```
for ( i = 0; i < 3; i ++ )  
{  
    a = a * 2;  
}
```

The diagram illustrates the components of a `for` loop. The loop header `for (i = 0; i < 3; i ++)` is shown with three parts highlighted in grey boxes: `i = 0;`, `i < 3;`, and `i ++`. Arrows point from labels to these parts: '초기식' (Initialization) points to `i = 0;`, '조건식' (Condition) points to `i < 3;`, and '증감식' (Increment) points to `i ++`. The loop body `{ a = a * 2; }` is shown with the statement `a = a * 2;` highlighted in a grey box. An arrow points from the label '실행문' (Execution Statement) to this statement.

for 문



for 문

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int a = 1;
```

```
    for (int i = 0; i < 3; i++) {
```

```
        a = a * 2;
```

```
    }
```

```
    printf("a = %d\n", a);
```

```
    getchar();
```

```
}
```

1. 초기식 - 변수 i 초기화. 딱 한 번 실행
2. 조건식을 검사하여 결과가 참이면 블록 안으로
3. 반복문 $a = a * 2$ 실행
4. 증감식으로 i값 1 증가
5. 다시 조건식 검사하여 참이면 블록 안으로, 거짓이면 빠져나옴

실행
결과 a : 8

for 문



```
for ( i = 0; i < 10; i++ )  
{  
    printf("Be happy!\n");  
}
```

```
i = 0;  
while ( i < 10 )  
{  
    printf("Be happy!\n");  
    i++;  
}
```

do~while 문

```
do  
{  
    a = a * 2; ← 반복할 문장  
} while (a < 10); ← 반복 조건
```

일단 반복할 문장 수행 후 조건은 나중에 검사

do~while 문

```
#include <stdio.h>

int main(void)
{
    int a = 1;

    do
    {
        a = a * 2;
    } while(a < 10);
    printf("a : %d\n", a);

    return 0;
}
```

반복문 중첩



중첩 반복문 예

```
for ( i=0; i<10; i++ )
{
    for ( j=0; j<10; j++ )
    {
        반복할 문장;
    }
}
```

i-for문이 10번 반복되고 j-for문이
10번 반복되므로 반복할 문장은
100번 반복된다.

분기문 사용 예

```
while ( 1 )
{
    if ( 조건식 1 ) break;
    if ( 조건식 2 ) continue;
    반복할 문장;
}
```

조건식 1이 참이면 반복문을 끝낸다.
조건식 2가 참이면 반복할 문장을
건너뛰고 처음부터 다시 반복한다.

반복문 중첩

```
int main(void)
{
    int i, j;

    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 5; j++)
        {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```

실행
결과

```
*****
*****
*****
```

반복문 활용 (구구단)



$$2 * 1 = 2$$

$$2 * 2 = 4$$

$$2 * 3 = 6$$

$$2 * 4 = 8$$

$$2 * 5 = 10$$

$$2 * 6 = 12$$

$$2 * 7 = 14$$

$$2 * 8 = 16$$

$$2 * 9 = 18$$

반복문 활용 (구구단)



```
for ( j = 1; j <= 9; j++ )  
{  
    printf("2 * %d = %d\n", j , 2 * j);  
}
```


반복문 활용 (구구단)

```
for ( i = 0; i < 8; i++ )           // 8번 반복
{
    for ( j = 1; j <= 9; j++ )
    {
        printf("2 * %d = %d\n", j, 2 * j); ← 2단만 출력
    }
}
```

반복문 활용 (구구단)

```
for ( i = 2; i <= 9; i++ )
{
    for ( j = 1; j <= 9; j++ )
    {
        printf("%d * %d = %d\n" , i, j, i * j ) ;
    }
}
```

반복문 제어 (break, continue)

```
int main(void)
{
    int i;                // 반복 횟수를 세기 위한 제어 변수
    int sum = 0;          // 1부터 10까지의 합을 누적할 변수


    for(i = 1; i <= 10; i++) // i는 1부터 10까지 증가하면서 10번 반복
    {
        sum += i;          // i값을 sum에 누적
        if(sum > 30) break; // 누적한 값이 30 보다크면 반복문을 끝낸다.
    }
    printf("누적한 값 : %d\n", sum);
    printf("마지막으로 더한 값 : %d\n", i);

    return 0;
}
```

실행 결과
누적한 값 : 36
마지막으로 더한 값 : 8

반복문 제어 (break, continue)

```
for ( ... )  
{  
    for ( ... )  
    {  
        ...  
        if ( 조건식 ) break;  
    }  
}
```



안쪽 for문 하나만 탈출

반복문 제어 (break, continue)

```
for ( i = 1; i <= 100; i++)  
{  
    if ( (i % 3) == 0 )  
    {  
        continue;  
    }  
    sum += i;  
}
```

i가 3의 배수면 `sum += i` 문장을 건너뛰고
블록 끝으로 간 후에 다시 반복한다.

무한 반복



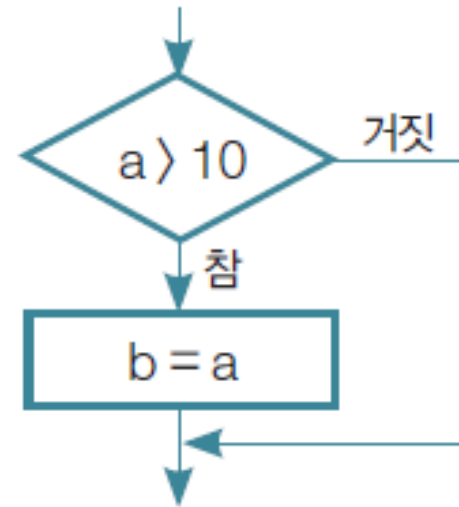
```
while ( 1 )  
{  
    printf("Be happy!\n");  
}
```

```
for ( ; ; )  
{  
    printf("Be happy!\n");  
}
```

if 문



```
if (a > 10)  → 조건식  
{  
    b = a;   → 실행문  
}
```



if 문



```
int main(void)
{
    int a = 20;
    int b = 0;

    if(a > 10)
    {
        b = a;
    }

    printf("a : %d, b : %d\n", a, b);

    return 0;
}
```

실행
결과 a : 20, b : 20

if 문



아래 코드는 모두 동일

```
if(a > b)
{
    b = a;
}
```

```
if(a > b) {
    b = a;
}
```

```
if(a > b)
    b = a;
```

```
if(a > b) b = a;
```

if 문



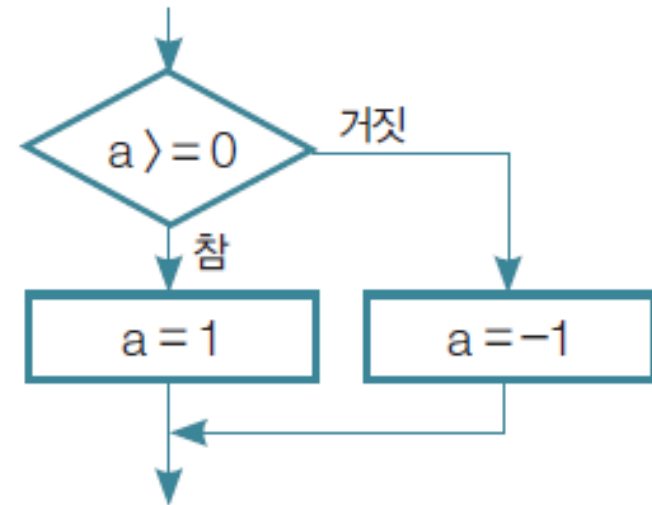
만일 {}로 묶지 않으면 **한 문장만 실행됨**. 따라서 여러 문장을 실행하려면 반드시 괄호로 묶어야 함.

```
if (a > 10)
    b = a;                // 여기까지가 if문
printf("a와 b는 같습니다."); // if문과 독립된 문장으로 항상 실행
```

if 문

if ~ else

```
if (a >= 0)  → 조건식  
{  
    a = 1;   → 실행문 1  
}  
else  
{  
    a = -1;  → 실행문 2  
}
```



if 문



```
int main(void)
{
    int a = 10;

    if(a >= 0)
    {
        a = 1;
    }
    else
    {
        a = -1;
    }

    printf("a : %d\n", a);

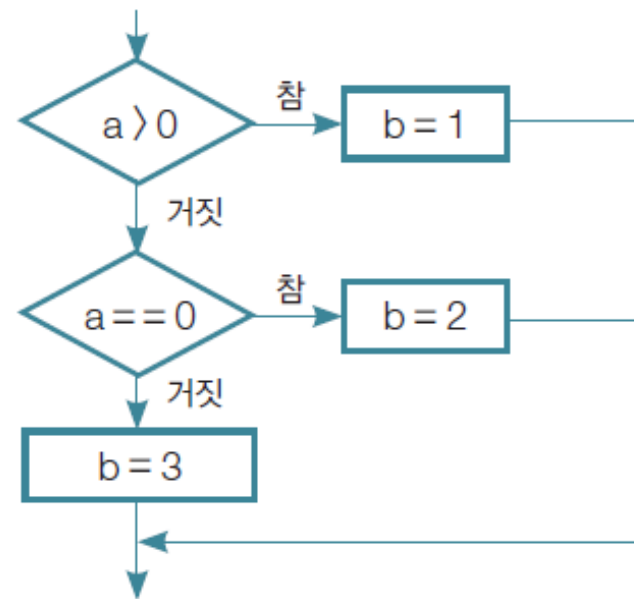
    return 0;
}
```

실행
결과 a : 1

if 문

if ~ else if ~ else

```
if(a > 0)      → 조건식 1
{
    b = 1;     → 실행문 1
}
else if(a == 0) → 조건식 2
{
    b = 2;     → 실행문 2
}
else
{
    b = 3;     → 실행문 3
}
```



if 문



```
int main(void)
{
    int a = 0, b = 0;

    if(a > 0)
    {
        b = 1;
    }
    else if(a == 0)
    {
        b = 2;
    }
    else
    {
        b = 3;
    }

    printf("b : %d\n", b);

    return 0;
}
```

실행
결과 b : 2

if 문



if문 안에 또 다른 if문 (중첩)

```
if ( 조건식 1 )  
{  
    if ( 조건식 2 )  
    {  
        실행문 1 ;  
    }  
    else  
    {  
        실행문 2 ;  
    }  
}
```

```
if ( 조건식 1 )  
{  
    if ( 조건식 2 )  
    {  
        실행문 1 ;  
    }  
    else  
    {  
        실행문 2 ;  
    }  
}
```

1부터 하나씩 검사

```
if(a == 1)
{
    printf("일");
}
else if(a == 2)
{
    printf("이");
}
else if(a == 3)
{
    printf("삼");
}
else if(a == 4)
{
    printf("사");
}
else if(a == 5)
{
    printf("오");
}
else
{
    printf("육");
}
```



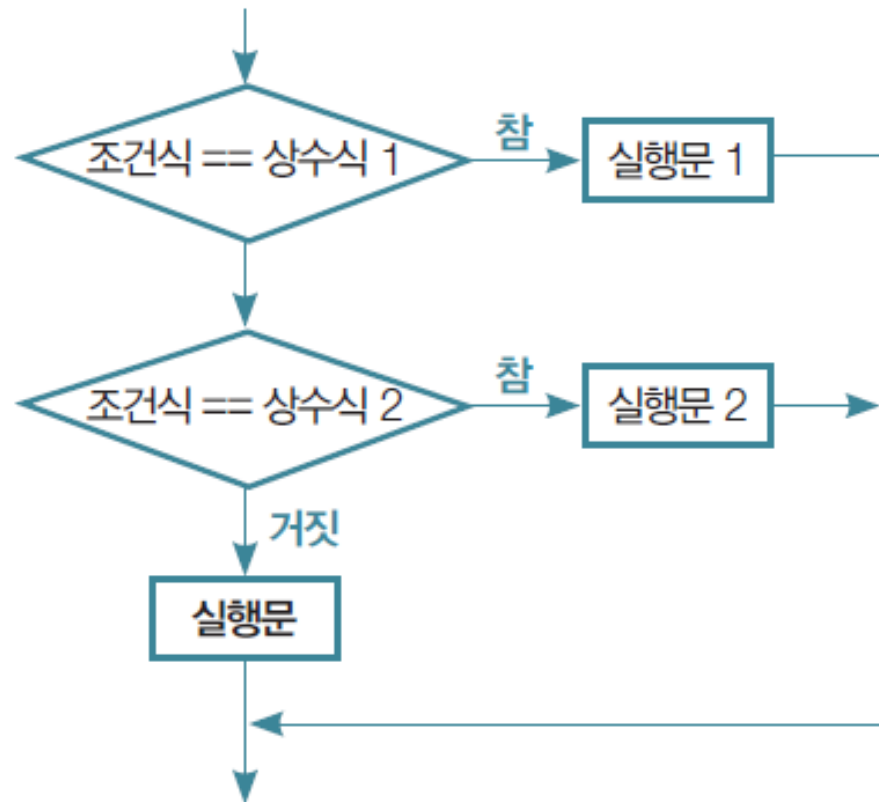
두 범위로 나누어 검사

```
if(a <= 3)
{
    if(a == 1)
    {
        printf("일");
    }
    else if(a == 2)
    {
        printf("이");
    }
    else
    {
        printf("삼");
    }
}
else
{
    if(a == 4)
    {
        printf("사");
    }
    else if(a == 5)
    {
        printf("오");
    }
    else
    {
        printf("육");
    }
}
```

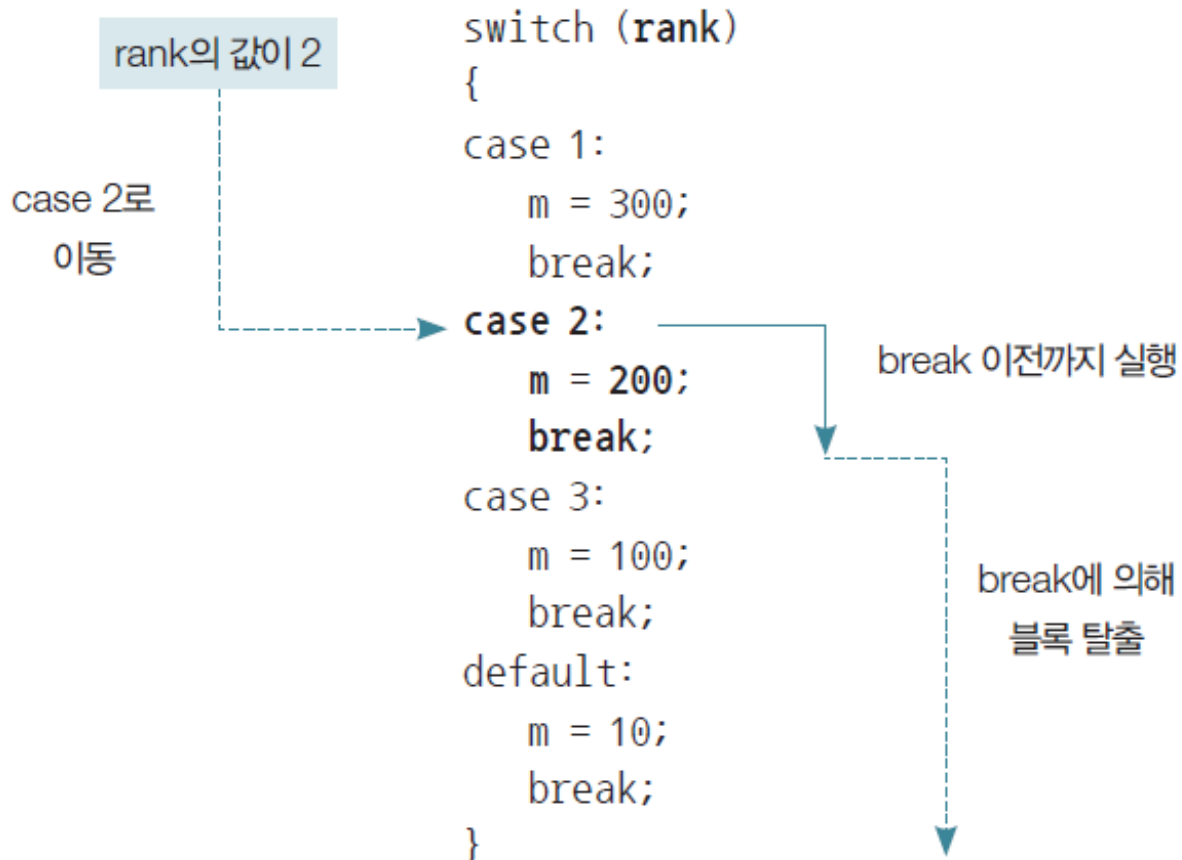
if문을 중첩해서 쓰는 이유
분할정복기법 : 실행 시간 단축

switch ~ case 문

```
switch ( 조건식 )  
{  
  case 상수식 1 :  
    실행문 1 ;  
    break ;  
  case 상수식 2 :  
    실행문 2 ;  
    break ;  
  default :  
    실행문 ;  
    break ;  
}
```



switch ~ case 문



switch ~ case 문과 if 문

```
switch (rank)
{
case 1:
    m = 300;
    break;
case 2:
    m = 200;
    break;
case 3:
    m = 100;
    break;
default:
    m = 10;
    break;
}
```



if문으로
바꿈

```
if (rank == 1)
{
    m = 300;
}
else if (rank == 2)
{
    m = 200;
}
else if (rank == 3)
{
    m = 100;
}
else
{
    m = 10;
}
```