C언어의 조건과 반복

01

순차/선택/반복 구조의 이해

02

선택 구조

1. if
2. switch
3. 번외: goto

03

반복 구조

1. for
2. while
3. do-while
4. 다중 for문

04

응용 문제

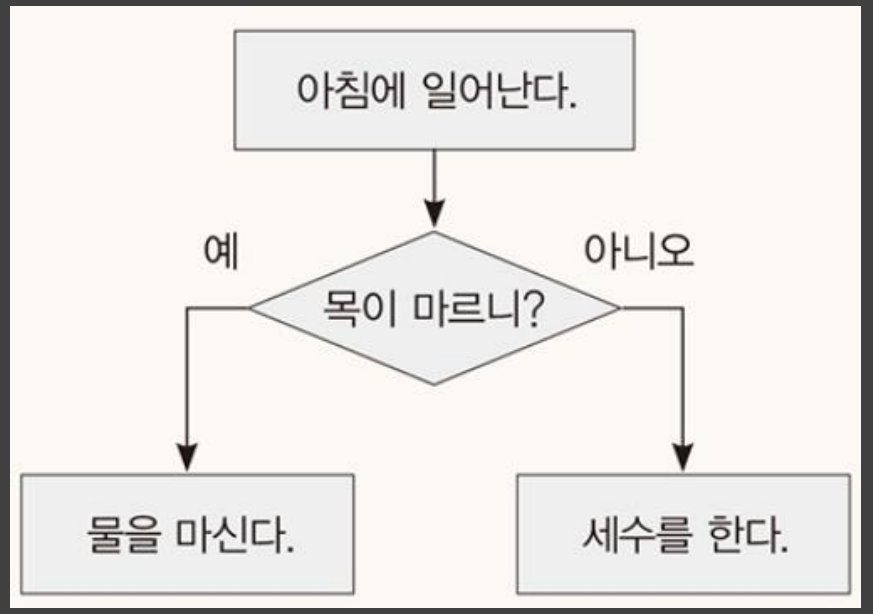
1. 계산기 만들기
2. 구구단 출력하기
3. 다이아몬드 별 찍기
4. 소인수분해하기

순차/선택/반복 구조

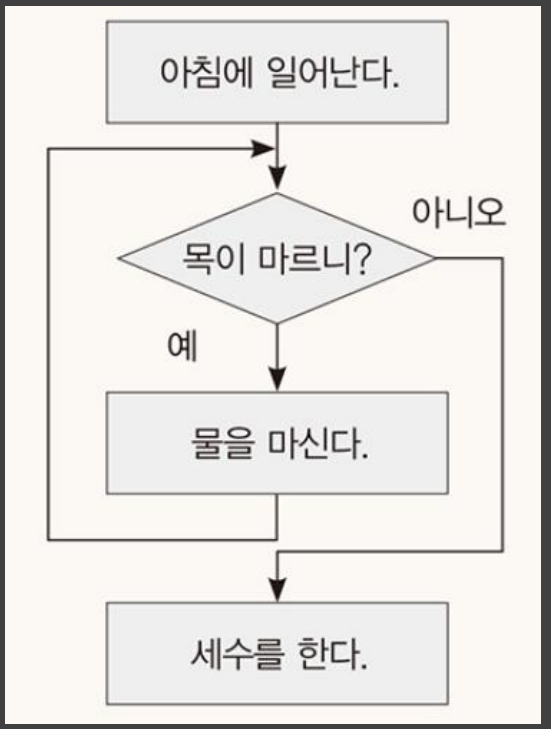
- **순차 구조:** 위에서 아래의 순서대로 실행하도록 명령을 내리는 구조
- **선택 구조:** 주어진 조건에 따라 명령을 선택적으로 실행하도록 명령을 내리는 구조
- **반복 구조:** 특정 횟수나 조건을 만족할 때까지 반복하도록 명령을 내리는 구조

순차/선택/반복 구조

순차 구조



선택 구조



반복 구조

이미지 출처: https://booksr.tistory.com/13

선택 구조: if

- **조건식이 참일 때 해당 statement를 수행하는 선택 구조**
- **if, else if, else 키워드가 있음**

1. if (조건식) statement;
2. if (조건식) statement;
   else statement;
3. if (조건식) statement;
   else if (조건식) statement;
   …
   else statement;

```
if (score >= 90) {
    printf("A입니다.");
} else if (score >= 80) {
    printf("B입니다.");
} else if (score >= 70) {
    printf("C입니다.");
} else if (score >= 60) {
    printf("D입니다.");
} else {
    printf("F입니다.");
}
```

※ statement

1. 한 줄로 쓸 경우 중괄호 생략 가능: if (조건식) statement;
2. 여러 줄로 쓸 경우 중괄호 필수: if (조건식) { statement1; statement2; }

선택 구조: if

- statement를 쓸 때 괄호 주의!

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80)
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    if (score >= 80)
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");

    return 0;
}
```

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80) {
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    }
    if (score >= 80) {
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");
    }

    return 0;
}
```

위 두 코드의 차이는?

선택 구조: if

- statement를 쓸 때 괄호 주의!

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80)
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    if (score >= 80)
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");

    return 0;
}
```

```
재수강을 할 수 있습니다.
B0 이상입니다.
재수강을 할 수 없습니다.
```

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80) {
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    }
    if (score >= 80) {
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");
    }

    return 0;
}
```

```
B0 이상입니다.
재수강을 할 수 없습니다.
```

선택 구조: if

- statement를 쓸 때 괄호 주의!

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80)
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    if (score >= 80)
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");

    return 0;
}
```

```c
#include <stdio.h>

int main() {
    int score = 85;

    if (score < 80) {
        printf("C+ 이하입니다.\n");
        printf("재수강을 할 수 있습니다.\n");
    }
    if (score >= 80) {
        printf("B0 이상입니다.\n");
        printf("재수강을 할 수 없습니다.\n");
    }

    return 0;
}
```

왼쪽은 if를 거치지 않는 부분이 있기 때문에 무조건 출력되었던 것이다.
statement에 여러 줄을 쓸 경우 반드시 중괄호로 묶어야한다!

선택 구조: if

```c
if (num == 7)
    printf("한 번 출력될까요\n");
if (num == 7)
    printf("두 번 출력될까요\n");
```

```c
if (num == 7)
    printf("한 번 출력될까요\n");
else if (num == 7)
    printf("두 번 출력될까요\n");
```

위 두 코드의 차이는?

선택 구조: if

- else는 '~가 아니라면'의 의미이다.

```c
if (num == 7)
    printf("한 번 출력될까요\n");
if (num == 7)
    printf("두 번 출력될까요\n");
```

```
한 번 출력될까요
두 번 출력될까요
```

```c
if (num == 7)
    printf("한 번 출력될까요\n");
else if (num == 7)
    printf("두 번 출력될까요\n");
```

```
한 번 출력될까요
```

C언어의 참/거짓 판별

- Java, Python 등 타 언어엔 참과 거짓 그 자체를 값으로 갖는 Boolean이란 타입이 있어, 이 값을 통해 참과 거짓을 판별한다.

```java
boolean bool1 = true;
boolean bool2 = false;
```

```java
if (true)
    System.out.println("참입니다.");
else
    System.out.println("거짓입니다.");
```

- 하지만 C언어에는 Boolean 타입이 없어 **조건식의 결과 그 자체로 판단하는데, 0인 경우에만 false로 취급하고, 그 외의 값인 경우엔 모두 true로 처리한다.**

```c
if (0)
    printf("참입니다.");
else
    printf("거짓입니다.");
```

거짓입니다.

```c
if (1)
    printf("참입니다.");
else
    printf("거짓입니다.");
```

참입니다.

```c
if (-1)
    printf("참입니다.");
else
    printf("거짓입니다.");
```

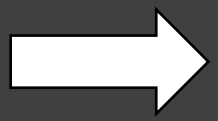
참입니다.

삼항/비교/논리 연산자

- **복습: 연산자**
  - **삼항연산자:** (조건식) ? 참일때수행 : 거짓일때수행;
  - **비교연산자:** ==, !=, >, <, >=, <=
  - **논리연산자:** ||, &&, !

- **연산의 결과는 거짓일 경우 0(false), 참일 경우 1(true)을 반환!**
- **if의 조건식에 다양한 연산자를 활용하여 코드를 간략화할 수 있다.**

```
If (score >= 60) {
    if (score <= 80) {
        statement;
    }
}
```



```
If (score >= 60 && score <= 80) {
    statement;
}
```

삼항/비교/논리 연산자

- **초보자들이 조건식을 쓸 때 흔히 하는 실수들**
  1. 60 <= score <= 80 (X)
     - 둘 이상의 비교 연산은 반드시 논리 연산자로 묶어야한다.
  2. score >= 60 & score <= 80 (X)
     - &는 논리 연산자가 아니라 비트 연산자이다.
  3. If (score = 60) (X)
     - =는 '같다'라는 비교 연산자가 아니라 대입 연산자이다.
     - 이 경우 반환값은 대입값인 60이 되고, 0이 아니므로 항상 참이 되는 조건식이 된다.

단축 평가

- **(조건1 || 조건2 …)에서 조건1이 True**
  - OR에선 하나만 참이어도 참이므로 뒤 조건 안따지고 바로 연산 종료
- **(조건1 && 조건2 …)에서 조건1이 False**
  - AND에선 하나만 거짓이어도 거짓이므로 뒤 조건 안따지고 바로 연산 종료

- **활용 예시: 음수 index로 배열 값 접근(예: arr[-1])을 방지하기 위해 바르게 쓴 것은?**

1. **if (index >= 0 && arr[index] >= 10) printf("%d", arr[index]);**
   - 만약 index값이 음수이면, 첫 조건부터 False가 되기 때문에 arr를 음수 인덱스로 접근하는 연산까지 하지 않아 의도치 않은 오류가 발생하지 않는다!

2. **if (arr[index] >= 10 && index >= 0) printf("%d", arr[index]);**
   - 만약 index값이 음수이면, 첫 조건에서 arr[음수]로 접근해버리기 때문에 의도치 않은 오류가 발생한다!

if 활용 예시: 0으로 나누기 방지

```c
#include <stdio.h>

int main() {
    double i, j;
    printf("나누고 싶은 두 수를 입력하세요: ");
    scanf("%lf %lf", &i, &j);

    if (j == 0) {
        printf("0으로 나눌 수 없습니다.\n");
        return 0;
    }
    printf("%f를 %f로 나눈 결과: %f\n", i, j, i / j);

    return 0;
}
```

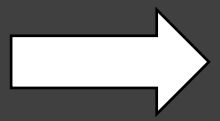
```
나누고 싶은 두 수를 입력하세요: 12 4
12.000000를 4.000000로 나눈 결과: 3.000000
```

```
나누고 싶은 두 수를 입력하세요: 12 0
0으로 나눌 수 없습니다.
```

선택 구조: switch

- 값의 case에 따라 해당 statement를 수행하는 선택 구조

```
If (score == 1) statement;
else if (score == 2)
statement;
...
else statement;
```



```
switch (score) {
    case 1:
        statement;
        break;
    case 2:
        statement;
        break;
    ...
    default:
        statement;
        break; // 생략 가능
```

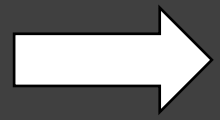
※ break 사용 주의!

break을 쓰지 않으면 '멈추지 않고' 계속 다음 코드를 쭉 수행한다.
원하는 동작만 수행시켜주기 위해 멈추게 하는 break 키워드를 써야한다!
break 키워드 자체에 대한 내용은 후술할 것.

선택 구조: switch

- 값의 case에 따라 해당 statement를 수행하는 선택 구조

```c
#include <stdio.h>

int main() {
    int n;
    printf("1: 결제하기, 0: 결제 취소하기");
    scanf("%d", &n);

    if (n == 0)
        printf("결제가 취소되었습니다.");
    else if (n == 1)
        printf("결제가 완료되었습니다.");
    else
        printf("알 수 없는 명령입니다.");
    return 0;
}
```



```c
int main() {
    int n;
    printf("1: 결제하기, 0: 결제 취소하기");
    scanf("%d", &n);

    switch (n) {
        case 0:
            printf("결제가 취소되었습니다.");
            break;
        case 1:
            printf("결제가 완료되었습니다.");
            break;
        default:
            printf("알 수 없는 명령입니다.");
            break;
    }
    return 0;
}
```

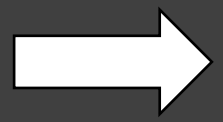
※ break 사용 주의!

break을 쓰지 않으면 '멈추지 않고' 계속 다음 코드를 쭉 수행한다.
원하는 동작만 수행시켜주기 위해 멈추게 하는 break 키워드를 써야한다!
break 키워드 자체에 대한 내용은 후술할 것.

선택 구조: switch

- **여러 case에 대해 같은 걸 수행하게도 할 수 있다.**

```
If (score == 1 || score == 2) statement;
else if (score == 3 || score == 4)
statement;
…
else statement;
```



```
switch (score) {
    case 1:
    case 2:
        statement;
        break;
    case 3:
    case 4:
        statement;
        break;
    …
    default:
        statement;
        break; // 생략 가능
```

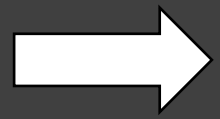
※ break 사용 주의!

break을 쓰지 않으면 '멈추지 않고' 계속 다음 코드를 쭉 수행한다.
원하는 동작만 수행시켜주기 위해 멈추게 하는 break 키워드를 써야한다!
break 키워드 자체에 대한 내용은 후술할 것.

선택 구조: switch

- 여러 case에 대해 같은 걸 수행하게도 할 수 있다.

```c
#include <stdio.h>

int main() {
    char n;
    printf("A: 결제하기, B: 결제 취소하기");
    scanf("%c", &n);

    if (n == 'B' || n == 'b')
        printf("결제가 취소되었습니다.");
    else if (n == 'A' || n == 'a')
        printf("결제가 완료되었습니다.");
    else
        printf("알 수 없는 명령입니다.");
    return 0;
}
```



```c
#include <stdio.h>

int main() {
    char n;
    printf("A: 결제하기, B: 결제 취소하기");
    scanf("%c", &n);

    switch (n) {
        case 'B':
        case 'b':
            printf("결제가 취소되었습니다.");
            break;
        case 'A':
        case 'a':
            printf("결제가 완료되었습니다.");
            break;
        default:
            printf("알 수 없는 명령입니다.");
            break;
    }
    return 0;
}
```

※ break 사용 주의!

break을 쓰지 않으면 '멈추지 않고' 계속 다음 코드를 쭉 수행한다.
원하는 동작만 수행시켜주기 위해 멈추게 하는 break 키워드를 써야한다!
break 키워드 자체에 대한 내용은 후술할 것.

선택 구조: switch

- 주의: case에 변수가 올 수 없음. 반드시 상수가 와야함!

```c
#include <stdio.h>

int main() {
    int n;
    printf("1: 결제하기, 0: 결제 취소하기");
    scanf("%d", &n);

    switch (n) {
        case n:
            printf("오류");
            break;
    }
    return 0;
}
```

번외: goto

- ‘goto 라벨이름’을 만날 경우, ‘라벨이름:’으로 점프하는 구조

```c
#include <stdio.h>

int main() {
    int answer;
    printf("C언어 강의를 구매하시겠습니까? (1: 예, 2: 아니오): ");
    scanf("%d", &answer);
    if (answer == 2) {
        goto Quit;
    } else {
        printf("1: 초급 과정 구매, 2: 중급 과정 구매, 3: 고급 과정 구매, 4: 구매 취소: ");
        scanf("%d", &answer);
        if (answer == 4) {
            printf("구매를 취소하셨습니다.\n");
        } else {
            printf("구매가 완료되었습니다.\n");
        }
    }

Quit:
    printf("강의 구매 프로그램을 종료합니다.\n");
    return 0;
}
```

번외: goto

- **goto의 단점**
  - **goto를 쓰면 프로그램의 흐름이 여기저기 왔다갔다하여 코드의 흐름 구조를 파악하기 힘들어진다.**
  - **함수의 스택을 날리고, 바로 해당 위치로 이동해버려 프로그램의 구조적 흐름이 꼬인다.**
    - 스파게티 면처럼 꼬인 코드라고 하여 스파게티 코드(Spaghetti code)라고 불림
    - 지양해야할 코딩 스타일이다.

번외: goto

- 그럼에도 goto를 쓰면 좋은 경우
  - 다중 반복 탈출하기
    - 물론 이 코드도 goto없이 구현이 가능하지만, 다중 반복 탈출 여부를 알 수 있는 변수를 추가해야하는 것이 번거로울 때 다음과 같이 응용할 수 있다.

```c
#include <stdio.h>

int main() {
    int quit = 0;

    for (int i = 1; i <= 9; i++) {
        printf("| ");
        for (int j = 1; j <= 9; j++) {
            if (i == 4 && j == 9) {
                quit = 1;
                break;
            }
            printf("%d * %d = %2d | ", i, j, i * j);
        }
        if (quit)
            break;
        printf("\n");
    }
    return 0;
}
```

goto를 사용하지 않고 구현

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 9; i++) {
        printf("| ");
        for (int j = 1; j <= 9; j++) {
            if (i == 4 && j == 9) {
                goto EXIT;
            }
            printf("%d * %d = %2d | ", i, j, i * j);
        }
        printf("\n");
    }
    EXIT:
    return 0;
}
```

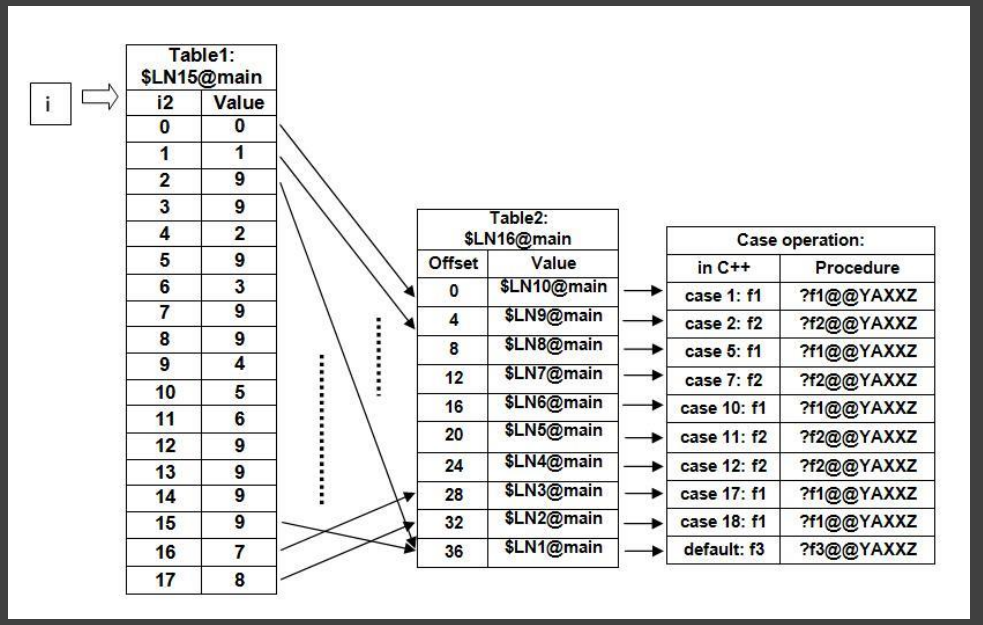
goto를 사용하여 구현

if, switch의 연산 속도 비교

- **if의 경우 조건마다 다 비교 연산을 수행한다.**
- **switch의 경우 컴파일러에 의해, 초기부터 case 값에 따라 어떤 것을 수행할 지가 미리 분기로 나뉜 jump table이란 것이 미리 생성된다.**
  - 그렇기 때문에 case의 값에는 변수가 올 수 없다. 미리 case 값을 컴파일러가 고정적으로 알고 있어야 jump table을 만들 수 있기 때문이다.
  - jump table을 미리 만들고, 들어온 값을 보고 바로 해당 분기로 JUMP하기 때문에 if처럼 분기마다의 연산이 필요하지 않다.

if, switch의 연산 속도 비교

- 요약
  - if는 조건을 만날때마다 검사하고, switch는 값만 받고 바로 해당 분기로 찾아가 수행
  - 그렇기에 연산 속도는 switch가 더 빠르다.
  - 또한 case 값의 크기가 작고, 값이 순차적으로 정렬되어있으며, 그 값 간의 차이가 크지 않을 수록 switch를 써서 연산 시간을 단축하는 효과가 극대화된다.



▲ jump table

반복 구조: for

- 초기식을 가지고 들어간 후, 조건식이 참인 동안 statement를 수행하고, 증감식을 수행한 후 조건식 검사를 반복하며 실행하는 구조

for (초기식; 조건식; 증감식) statement;

- 초기식: for 들어가기 전 딱 한 번만 수행됨
- 조건식: 조건식이 참인 동안 반복 수행됨
- 증감식: 반복 끝날때마다 해당 부분이 한 번 수행됨

```
for (int i = 0; i < 10; i++)
    printf("%d ", i);
```

```
0 1 2 3 4 5 6 7 8 9
```

for는 흔히 반복횟수가 명확할 때 사용한다.

반복 구조: while

- 조건식이 참인 동안 statement를 수행하고, 한 번 수행이 끝날때마다 다시 조건식 검사를 반복하며 실행하는 구조

while (조건식) statement;

- 조건식: 조건식이 참인 동안 반복 수행됨

```c
int i = 0;
while (i < 10) {
    printf("%d ", i);
    i++;
}
```

```
0 1 2 3 4 5 6 7 8 9
```

while은 흔히 반복횟수가 명확하지 않을 때 사용한다.

반복 구조: do-while

- while과 똑같이 조건식이 참인 동안 반복 수행되지만, 맨 처음 한 번의 statement 실행은 조건식 검사 없이 보장하는 반복 구조

```
do
    statement;
while (조건식);
```

- 조건식: 조건식이 참인 동안 반복 수행됨

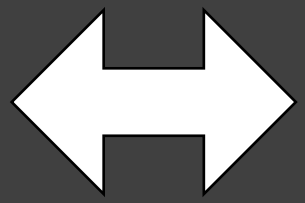
```c
do {
    printf("처음 한 번은 무조건 실행");
} while (0);
```

```
처음 한 번은 무조건 실행
```

반복 구조 비교

- for와 while을 서로 바꿔써보며 둘의 차이를 이해하기
  - 1~100까지 출력하기

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++)
        printf("%d ", i);
    return 0;
}
```



```c
#include <stdio.h>

int main() {
    int i = 1;
    while (i <= 100) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

for (초기식; 조건식; 증감식) { statement; }

초기식;
while (조건식) { statement; 증감식; }

무한 반복

- 조건식이 항상 참이면 무한반복이 수행된다.

for(;;) statement;

- 초기식, 조건식, 증감식 모두 없음.
- 이때 조건은 항상 참이고, 증감식에 의해 조건의 참/거짓이 달라질리도 없어 무한히 수행됨

```c
#include <stdio.h>

int main() {
    for (;;)
        printf("NEVER ENDING ");
    return 0;
}
```

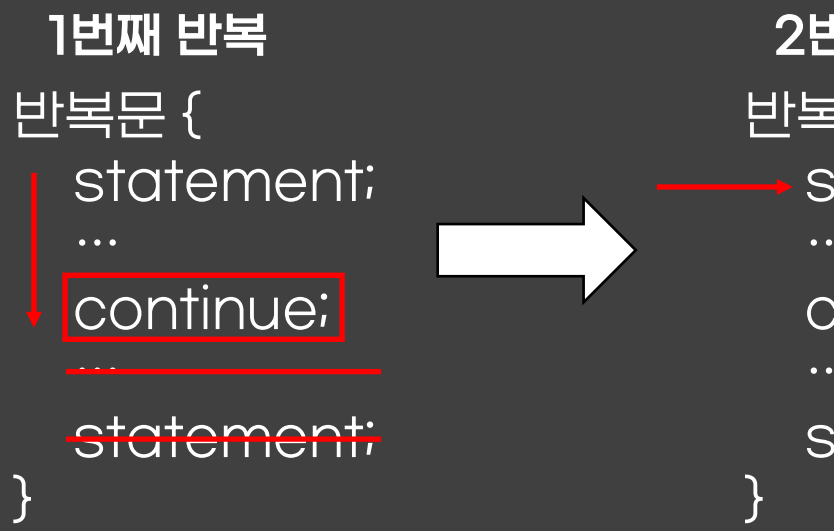
while (1) statement;

- 조건식 부분에 0이 아닌 값, 즉 참인 값이 들어가있어 항상 참이 되므로 무한히 수행됨

```c
#include <stdio.h>

int main() {
    while (1)
        printf("NEVER ENDING ");
    return 0;
}
```

반복 흐름 제어: continue

- 반복 수행 중 continue를 만나면, 그 밑의 모든 코드 수행을 건너뛰고 다음 반복을 수행한다.



반복 흐름 제어: continue

- 반복 수행 중 continue를 만나면, 그 밑의 모든 코드 수행을 건너뛰고 다음 반복을 수행한다.

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i % 5 == 0)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        printf("%d ", i);
        if (i % 5 == 0)
            continue;
    }
    return 0;
}
```

1부터 100까지 출력하되 5의 배수는 **건너뛰고** 모두 출력하는 코드로 올바른 것은?

반복 흐름 제어: continue

- 반복 수행 중 continue를 만나면, 그 밑의 모든 코드 수행을 건너뛰고 다음 반복을 수행한다.

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i % 5 == 0)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

```
1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19 21 22 23 24 26 27 28 29 31 32 33
 34 36 37 38 39 41 42 43 44 46 47 48 49 51 52 53 54 56 57 58 59 61 62 63
 64 66 67 68 69 71 72 73 74 76 77 78 79 81 82 83 84 86 87 88 89 91 92 93
 94 96 97 98 99 * 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키
```

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        printf("%d ", i);
        if (i % 5 == 0)
            continue;
    }
    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 * 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

반복 흐름 제어: continue

- 반복 수행 중 continue를 만나면, 그 밑의 모든 코드 수행을 건너뛰고 다음 반복을 수행한다.

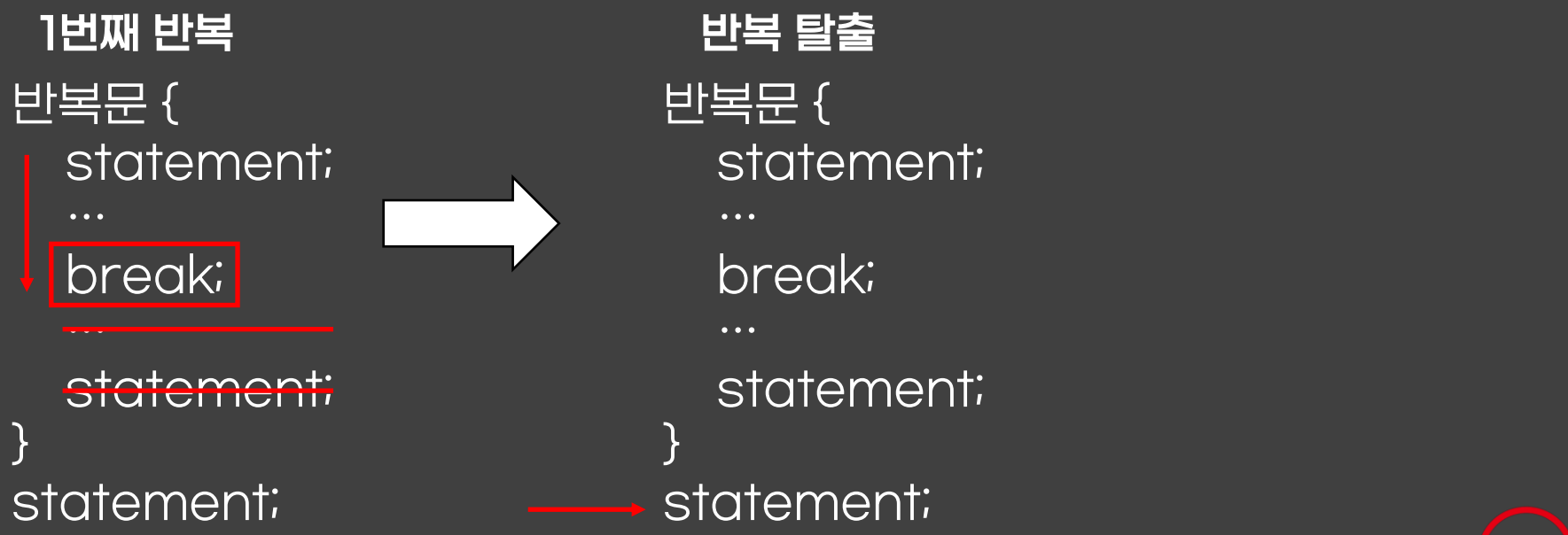
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i % 5 == 0)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

1부터 100까지 출력하되 5의 배수는 **건너뛰고** 모두 출력하기

If를 먼저 쓰는 것과, printf를 먼저 쓰는 것과의 결과 차이에 주의!
프로그램의 흐름을 잘 보아야한다!

반복 흐름 제어: break

- 반복 수행 중 break을 만나면, 그 밑의 모든 코드 수행을 무시하고 즉시 반복을 탈출한다.



반복 흐름 제어: break

- 반복 수행 중 break을 만나면, 그 밑의 모든 코드 수행을 무시하고 즉시 반복을 탈출한다.

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i == 50)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        printf("%d ", i);
        if (i == 50)
            break;
    }
    return 0;
}
```

1부터 100까지 출력을 시도하되 50을 만나는 순간 출력하지 않고 **반복 중단**하는 코드로 올바른 것은?

반복 흐름 제어: break

- 반복 수행 중 break을 만나면, 그 밑의 모든 코드 수행을 무시하고 즉시 반복을 탈출한다.

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i == 50)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
```

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        printf("%d ", i);
        if (i == 50)
            break;
    }
    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

반복 흐름 제어: break

- 반복 수행 중 break을 만나면, 그 밑의 모든 코드 수행을 무시하고 즉시 반복을 탈출한다.

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        if (i == 50)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 100; i++) {
        printf("%d ", i);
        if (i == 50)
            break;
    }
    return 0;
}
```

1부터 100까지 출력을 시도하되 50을 만나는 순간 출력하지 않고 반복 중단하기

If를 먼저 쓰는 것과, printf를 먼저 쓰는 것과의 결과 차이에 주의!
프로그램의 흐름을 잘 보아야한다!

이중/다중 for문

- **for문 안에 for문이 들어있는 구조**
  - for문이 이중으로 겹쳐있으면 이중 for문, 그보다 많이 겹쳐있으면 다중 for문이라고 한다.
  - 주로 규칙성이 있는 2차원이나 그 이상 n차원 데이터 값들을 다룰 때 많이 활용한다.

```
for () {
   for () {
      …
   }
}
```

```c
#include <stdio.h>

int main() {
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            printf("(%d, %d)\n", i, j);
        }
    }
    return 0;
}
```

```
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(0, 4)
(0, 5)
(0, 6)
(0, 7)
(0, 8)
(0, 9)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(1, 4)
(1, 5)
```

…

```
(8, 4)
(8, 5)
(8, 6)
(8, 7)
(8, 8)
(8, 9)
(9, 0)
(9, 1)
(9, 2)
(9, 3)
(9, 4)
(9, 5)
(9, 6)
(9, 7)
(9, 8)
(9, 9)
```

좌표 (0, 0), (0, 1), …부터 …, (9, 8), (9, 9)까지 모두 출력하기

응용 문제

- **if/switch를 활용한 계산기 만들기**
  - 두 정수와 연산자를 입력받은 후, 선택 구조인 if/switch를 이용하여 연산자에 따라 더하기/빼기/곱하기/나누기/나머지 연산을 수행하는 계산기를 만들어라. 만약 올바르지 않은 연산자가 입력될 경우, "올바르지 않은 연산자입니다"가 출력되도록 하라. 주어진 makeACalculatorSwitch.c의 스켈레톤 코드를 완성하여 다음 결과 출력 예시처럼 올바른 계산 기능을 수행할 수 있도록 하라. 이때 makeACalculatorSwitch.c 외 다른 파일은 편집할 수 없으며 추가적으로 파일을 생성해선 안된다.

응용 문제

- **for문을 활용한 구구단 출력하기**
  - 반복 구조인 for를 이용하여 1 * 1부터 9 * 9까지 모든 구구단을 출력하라. 주어진 printMultiplicationTable.c의 스켈레톤 코드를 완성하여 다음 결과 출력 예시처럼 올바른 구구단을 출력할 수 있도록 하라. 이때 printMultiplicationTable.c 외 다른 파일은 편집할 수 없으며 추가적으로 파일을 생성해선 안된다.

응용 문제

- **다이아몬드 별 찍기**
  - n값을 입력받은 후, 반복 구조를 이용하여 2 * n + 1 (n >= 0) 크기의 별 다이아몬드를 출력하라. 주어진 printStarDiamond.c의 스켈레톤 코드를 완성하여 다음 결과 출력 예시처럼 올바른 크기의 별 다이아몬드를 출력할 수 있도록 하라. 이때 printStarDiamond.c 외 다른 파일은 편집할 수 없으며 추가적으로 파일을 생성해선 안된다.

응용 문제

- **소인수분해하기**
  - n값을 입력받은 후, 선택 구조와 반복 구조를 이용하여 n값에 대한 소인수분해 결과를 출력하라. 주어진 primeFactorization.c의 스켈레톤 코드를 완성하여 다음 결과 출력 예시처럼 올바른 소인수분해 결과를 출력할 수 있도록 하라. 이때 primeFactorization.c 외 다른 파일은 편집할 수 없으며 추가적으로 파일을 생성해선 안된다.

THANK YOU!