Control Flow

adopted from KNK C Programming : A Modern Approach

Iteration Statements (반복 문장)

- C의 반복문은 루프(명령들의 반복사용)를 만들 때 사용됨.
- ***loop*** (루프) 는 여러 문장을 반복 실행할 때 쓰임 (the ***loop body*** ***루프 바디에 반복할 문장 작성***)

- C의 모든 루프는 ***controlling expression*** 제어식이 필요
  - 루프의 바디가 실행될 때마다 (루프 한 번 회전), 제어식이 재평가 됨
  - 제어식이 참일 때만 (0이 아닐 때만) 루프는 계속 반복함

Iteration Statements

- C 에는 3 종류의 반복문이 있음:
  - `while` 문은 루프 바디 보다 제어식이 먼저 평가되야 할 때 사용
    제어구문 평가 후 참이면 루프바디 실행
  - `do` 문은 루프 바디가 제어식 보다 먼저 실행되야 할 때 사용.
    루프바디 1회 실행후 제어구문 평가, 참이면 다시 실행
  - `for` 문은 카운트 용 변수를 증가 감소 하는 경우 사용
    변수의 증감이 쉬운 반복문

The `while` Statement

- `while` 문은 가장 쉽게 만들 수 있는 반복문
- `while` 문의 형식

  `while` (*expression*) *statement*

- 표현식이 제어식이고 루프 바디에는 문장이 있음

The `while` Statement

- 예:

```
while (i < n)   /* 제어식 */
  i = i * 2;    /* loop body */
```

- `while` 문이 실행되면, 제일 먼저 제어식을 평가함
- 0이 아니면 /true 이면 루프바디가 실행되고, 다시 제어 구문을 평가함
- 제어식의 값이 0이 되거나 false 가 될 때까지 반복함

The `while` Statement

- `while` 문으로 n보다 크거나 같은 가장 작은 2의 배수를 찾는 문장:

```
i = 1;
while (i < n)
  i = i * 2;
```

- 루프의 실행 조건 트레이스 `n` = 10:

`i = 1;`	`i` 는 1.
`i < n` 참인가?	참; 계속.
`i = i * 2;`	`i` 는 2.
`i < n` 참인가?	참; 계속.
`i = i * 2;`	`i` 는 4.
`i < n` 참인가?	참; 계속.
`i = i * 2;`	`i` 는 8.
`i < n` 참인가?	참; 계속.
`i = i * 2;`	`i` 는 16.
`i < n` 참인가?	거짓; 루프 종료.

The `while` Statement

- 여러 문장이 실행되어야 하면 괄호를 사용함

```
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

- 엄격하게 필요하지 않더라도 항상 괄호를 쓰기도 함

```
while (i < n) {
  i = i * 2;
}
```

The `while` Statement

- 아래의 문장들은 카운트 다운 메시지를 출력함:

```
i = 10;                    //i가 0 이하일 때만 i>0이 거짓
while (i > 0) {            //0이하이면 루프바디가 실행안됨
  printf("T minus %d and counting\n", i);
  i--;                     // 다음처럼 한 줄로 표현 가능
}                          // printf("T minus %d and counting\n", i--);
```

- 최종적으로 출력되는 메시지는 `T minus 1 and counting`.

Infinite Loops 무한 루프

- `while` 평가 결과가가 0이외의 값이면 종료 안함
- 0이 아닌 상수를 써서 일부러 무한 루프를 만들기도 함

```
while (1) …
```

- 이런 종류의 `while` 문은 루프 바디 내에 루프 밖으로 나갈 수 있는 문장 (`break`, `goto`, `return`)이나 함수를 사용해야 함

The do Statement

- 일반적인 do 문의 형식

  do *statement* while (*expression*) ;

- do 문장이 실행되면 루프 바디가 먼저 한 번 실행되고, 제어식을 평가하여 다시 실행할지 여부를 결정

- "제어구문이 0이 아니면 루프바디를 재 실행하고 다시 제어구문을 평가"의 반복

The do Statement

- 카운트 다운 예제를 do 문을 다시 작성함:

```
i = 10;
do {                    // do, while에서는 항상 중괄호 사용
  printf("T minus %d and counting\n", --i);
} while (i > 0);
```

- do 와 while 문 대체적으로 차이가 없음
  - 유일한 차이는 최소 1번 루프바디 실행된다는 것

The `for` Statement

- `for` 문은 카운팅 용 변수가 있을 때 유용함
- 그 외에도 다양한 방식으로 루프를 구현할 수 있음

- 일반적인 `for` 문 형식:

초기 값 묶음 제어구문 묶음 증가구문 묶음

`for ( expr1 ; expr2 ; expr3 ) statement`

expr1, *expr2*, *expr3* 는 표현식

- Example:

```
for (i = 10; i > 0; i--)
  printf("T minus %d and counting\n", i);
```

The `for` Statement

- 예외적인 경우를 제외하면 `for` 은 언제나 `while` 로 변환 가능함

```
expr1;
while ( expr2 ) {
    statement
    expr3;
}
```

The `for` Statement

- *expr1* 는 루프 바디가 실행되기 전 딱 한 번만 초기화 되
- *expr2* 는 루프 반복의 종료를 제어 함 0이 아니면 계속 실행, 심지어 없어도 실행
- *expr3* 는 루프 바디가 한 번 실행된 후 실행됨

변환

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  i--;
}
```

```
for (i = 10; i > 0; i--)
  printf("T minus %d and counting\n", i);
```

The `for` Statement

- `while` 문으로 표현된 것과 비교해보면 `for` 문 이해에 도움됨

- 아래 예에서 `i--` 가 `--i`으로 바뀌면 어떻게 될까?

```
for (i = 10; i > 0; --i)
  printf("T minus %d and counting\n", i);
```

- `while` 루프로 표현한 경우 동작에 차이가 없음:

```
i = 10;
while (i > 0) {
  printf("T minus %d and counting\n", i);
  --i;
}
```

for-to-while.c

```
/* case 1 */
    for (int i = 10; i > 0; --i)
        printf("T minus %d and counting\n", i);

/* case 2 */
    printf("\n\ncase 2\n");
    for (int i = 10; i > 0; i--)
        printf("T minus %d and counting\n", i);

/* case 3 */
    printf("\n\ncase 3\n");
    int i = 10;
    while (i > 0) {
        printf("T minus %d and counting\n", --i);
     // printf("T minus %d and counting\n", i--);
```

실행 결과의 차이를 확인해보자

The `for` Statement

- `for` 문에서 첫 수식과 마지막 수식의 값은 중요하지 않고 그 문장으로 인한 영향만 의미 있음

- 결과적으로, 첫 수식과 마지막 수식은 할당이나 증감 관련 수식만 쓰임

for 문장의 숙어적 용법

- for 문은 대체적으로 변수의 값을 반복적으로 더하거나 또는 빼는 루프에서 유용
- for 문으로 n 번 더하거나 빼는 패턴은 다음과 같음

Counting up from* 0 *to* n–1*: `for (i = 0; i < n; i++) …`

Counting up from* 1 *to* n*: `for (i = 1; i <= n; i++) …`

Counting down from* n–1 *to* 0*: `for (i = n - 1; i >= 0; i--) …`

Counting down from* n *to* 1*: `for (i = n; i > 0; i--) …`

`for` Statement Idioms

- 흔한 `for` 문자에서의 실수:
  - 제어식에 `<`을 쓰는 대신 `>` (또는 반대로) 쓰는 경우. "더하기" 는 `<` 또는 `<=` 연산자를, "빼기"는 `>` 또는 `>=` 을 써야 함
  - 제어식에 `==` 을 쓰는 대신 `<`, `<=`, `>`, `>=` 을 쓰는 경우.
  - "Off-by-one" 하나차이 오류 `i <= n` 대신 `i < n` 쓴 경우.

for 문에서 표현식 안쓰기

- C 에서는 표현식을 모두 또는 일부를 안쓰는 것을 허용

- 첫 표현식이 없으면 초기화가 없는 것

```
i = 10;
for (; i > 0; --i)
  printf("T minus %d and counting\n", i);
```

Omitting Expressions in a `for` Statement

- 세 번째 표현식이 없으면 증감조건이 없음. 루프 바디에서 증감을 하여 종료 조건을 만족시켜야 함

```
for (i = 10; i > 0;  )
  printf("T minus %d and counting\n", i--);
```

Omitting Expressions in a `for` Statement

- 첫 번째와 마지막 표현식이 없는 경우는 while문과 같음

```
for (; i > 0;)
  printf("T minus %d and counting\n", i--);
```

는 아래와 같음

```
while (i > 0)
  printf("T minus %d and counting\n", i--);
```

- `while` 의 경우가 더 읽기 좋음

Omitting Expressions in a `for` Statement

- 두 번째 표현식이 없으면 항상 참으로 판단하여 `for` 문 종료하지 않음.
  - 루프 바디에서 종료 처리를 하지 않으면 무한 반복

- 무한 루프의 표현 방법 중 하나

```
for (;;) …
```

C99 표준에서 `for`문

- C99 문에서는 첫 번째 수식에 선언문을 쓸 수 있음
- 루프 내에서만 사용할 변수를 선언할 수 있도록 함:

```
for (int i = 0; i < n; i++)
  …
```

- 변수 `i` 는 그 이전에 선언 된 적이 없어도 이렇게 쓸 수 있음

`for` Statements in C99

- `for` 문에서 선언된 변수는 루프 바디 밖에서 보이지 않음

```
for (int i = 0; i < n; i++) {
  …
  printf("%d", i);
   /* legal; i is visible inside loop */
  …
}
printf("%d", i);    /*** WRONG ***/
```

`for` Statements in C99

- `for` 문에서만 쓰일 제어 용 변수를 쓰면 코드를 이해하는데 도움을 줌
- 만약 루프 종료 후에도 변수를 써야 하면 밖에서 선언해야 함

- `for` 문에서 여러 변수를 동시에 선언할 수 있음. 단 같은 형만

```
for (int i = 0, j = 0; i < n; i++)
  …
```

The Comma Operator

- 경우에 따라 하나 이상의 변수를 초기화하고 증감해야 할 수 있음

- 첫 번째와 마지막 수식에 ***comma expression 쉼표 연산자***를 써서 변수 추가 가능

- 다음의 형식을 따름

  expr1 , expr2

  expr1 와 *expr2* 는 두 개의 표현식

The Comma Operator

- *expr1* 에는 side effect가 있어야 의미있음

- 쉼표 연산자로 `++i, i + j`가 계산되면 `i` 가 먼저 증가되고 `i + j` 가 계산됨. i+j 계산시 증가한 i를 활용
  - `i` 와 `j` 가 최초 1 과 5 였다면 식의 결과는7. 그리고 `i` 는 2.

The Comma Operator

- 쉼표 연산자는 왼쪽 결합이므로 컴파일러는 다음을

  ```
  i = 1, j = 2, k = i + j
  ```

  아래와 같이 평가함

  ```
  ((i = 1), (j = 2)), (k = (i + j))
  ```

  왼쪽에서 오른쪽으로 식이 결합됨

The Comma Operator

- 쉼표 연산자는 두개의 식을 하나의 수식처럼 연결함
- Example:

```
for (sum = 0, i = 1; i <= N; i++)
  sum += i;
```

- 쉼표로 둘 이상의 변수 선언 가능

루프 밖으로 나가기

- `while` 또는 `for` 문장의 일반적인 종료 시점은 제어식이 있는 시작 부분이고 `do` 문장의 경우 마지막 부분임

- break를 사용하여 루프바디 중간 또는 임의의 지점에서 종료할 수 있음

The `break` Statement

- `break` 문은 switch문외에 while, do, for 문에서도 break 사용 가능하며 루프를 종료 할 때 쓰임

- 예: n이 소수인지 판단하는 루프에서, 소수를 찾으면 break로 즉시 종료시킬 수 있음

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;
```

The `break` Statement

- `break` 문으로 루프가 종료된 이후 결과를 확인해서 너무 일찍 종료되지 않았는지 검증해야 함

```
if (d < n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

The `break` Statement

- `break` 문은 루프 바디 실행 중간에 종료해야 하는 시점이 있는 경우 유용함

- 사용자 입력을 읽어서 특정 조건의 값이 입력되었는지 검사하는 예

```
for (;;) {
  printf("Enter a number (enter 0 to stop): ");
  scanf("%d", &n);
  if (n == 0)
    break;
  printf("%d cubed is %d\n", n, n * n * n);
}
```

The `break` Statement

- `break` 문은 `while`, `do`, `for`, `switch`의 중괄호 묶음 하나 밖으로 나가는 역할만 함
- 중첩 된 경우 하나의 묶음만 종료됨
- Example:

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

- `break` 문은 `switch` 문만 탈출하고`while` 루프는 계속 실행됨

The `continue` Statement

- `continue` 는 `break`와 유사함
  - `break` 는 제어권을 루프 밖으로 이동 시킴
  - `continue` 는 제어권을 루프 내에서 마지막 줄 끝으로 이동

- `break`는 루프 밖의 문장이 제어권 획득; `continue`는 제어권이 루프내에 유지
- `break` 와 `continue`의 또 다른 차이: `break` 는 `switch` 문과 반복문에 (`while`, `do`, `for`) 쓸 수 있지만, `continue`는 루프에만 쓰임

The `continue` Statement

- `continue` 문을 쓰는 루프:

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

The `continue` Statement

- `continue` 없는 앞의 예:

```
n = 0;
sum = 0;
while (n < 10) {
  scanf("%d", &i);
  if (i != 0) {
    sum += i;
    n++;
  }
}
```

The goto Statement

- goto 문은 ***label***이 있는 어디로든 이동 가능

- 라벨은 실행될 문장 앞에 있는 식별자임:

  라벨 실행될 문장
  identifier : *statement*

- Goto문은 다음과 같은 형식을 갖음

  라벨
  goto *identifier* ;

- goto *L*; 을 실행하면 라벨*L*이 있는 곳으로 실행 흐름을 바꿈. 단, 라벨은 goto 와 같은 함수 내에 있어야 함

The goto Statement

- C에 break 문이 없었다면 goto문을 이용해 루프 밖으로 나올 수 있음

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    goto done;

done:
if (d < n)
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

The goto Statement

- goto는 break, continue, return, exit등으로 대부분 대체될 수 있기에 거의 안쓰임
- 중첩된 반복문 내에 switch가 있는 경우 루프에서 빠져 나올 경우에 활용됨
- break 로는 원하는 효과를 얻을 수 없음. Switch에서는 나오겠지만 while 문 밖으로는 못나감
- goto 문을 쓰면 해결됨:

```
while (…) {
  switch (…) {
    …
    goto loop_done;    /* break won't work here */
    …
  }
}
loop_done: …
```

- goto 문은 중첩 루프 밖으로 나오는 데 유용함

The Null Statement

- 루프바디의 실행될 문장이 null일 수 있음, null은 비어 있다는 뜻
- 다음은 세 개의 문장으로 구성됨:

```
i = 0; ; j = 1;
```

세미콜론 수로 문장이 3개 있는 것을 알 수 있음. 단, 2번 문장은 어떤 일도 안함

- null문장을 쓰는 이유는 루프 내에서 할 일이 없을 때를 표현하기 위함

The Null Statement

- 다음의 예는 소수 찾는 루프임:

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;
```

앞에서 본 소수를 찾는 문장에 if 절의 수식(n%d == 0)을 for 의 종료 조건에 넣으면 루프 내에서 할 일이 없음

- `n % d == 0` 을 제어식으로 이동하면 루프 내에서 할 일이 없음:

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

  /* empty loop body */ ;
```

- 반복문을 독립 문장으로 쓸 때 혼란을 줄이기 위해 널 문장을 한 줄에 독립적으로 기록함.

The Null Statement

- 주의: 세미콜론을 `if`, `while`, `for` 뒤에 붙이면 널 문장을 삽입한 것
- Example 1:

```
if (d == 0);                                    /*** WRONG ***/
  printf("Error: Division by zero\n"); //if와 상관없이 출력
```

`printf` 의 호출은 `if` 문 내에 있는 것이 아니기 때문에 `d` 와 상관없이 실행됨

- Example 2:

```
i = 10;
while (i > 0);                                  /*** WRONG ***/
{
  printf("T minus %d and counting\n", i); //while과 상관없이 출력
  --i;                                    //while과 상관없이 출력
}
```

이 경우 무한 루프가 됨

The Null Statement

- Example 3: while은 루프바디 없이 1회 실행하고 printf문장이 실행됨, 들여 쓰기에 속으면 안됨

```
i = 11;
while (--i > 0);                    /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```

루프 바디는 Null이기 때문에 결과는:

```
T minus 0 and counting
```

- Example 4: for 문으로 작성 된 것 외에는 위의 예제와 동일

```
for (i = 10; i > 0; i--);           /*** WRONG ***/
  printf("T minus %d and counting\n", i);
```