Selection Statment

adopted from KNK C Programming : A Modern Approach

Statements 문장

- 지금까지 배운 것은 `return` 문장과 표현식 문장
- C의 나머지 문장들은 아래 3개 부류로 구분됨:
  - ***Selection statements 선택문:*** `if` 와 `switch`
  - ***Iteration statements 반복문:*** `while`, `do`, 와 `for`
  - ***Jump statements 점프문:*** `break`, `continue`, 와 `goto`. (`return` 는 이 부류에 속함.)
- 다른 C 문장들:
  - 합성 문장
  - Null 문장

Logical Expressions 논리 표현식

- C의 여러 문장은 “**true**” or “**false**”를 알기 위해 문장을 평가함
- 예를 `if` 문장이 `i < j`를 검사함; 이 문장이 참이면 `i`은 `j`보다 작다는 뜻.
- `i < j`의 비교 결과는 정수를 갖음: **0 (false) 또는 1 (true).**

Relational Operators 관계 연산자

- C의 ***relational operators 관계 연산자:***

  `<` less than
  `>` greater than
  `<=` less than or equal to
  `>=` greater than or equal to

표현식에 쓰이면 0 (false) , 1 (true)의 결과를 갖음

- 관계 연산자의 우선 순위는 수식 연산자들보다 낮음
  - example, `i + j < k - 1` 는 `(i + j) < (k - 1)`을 뜻함.
- 관계 연산자는 왼쪽에서 오른쪽으로 결합됨.

```
i < j < k
```

```
(i < j) < k
```

`i < j` 의 결과로 1 또는 0 가 나오고 그 후에 `k`와 비교됨

Equality Operators 동치 연산자

- C 에는 두개의 ***equality operators 동치 연산자가 있음:***

  `==` equal to

  `!=` not equal to

  왼쪽 결함

  결과로 0 (false) 또는 1 (true)

- 동치 연산자는 관계 연산자보다 우선순위가 낮음

```
i < j == j < k
```

```
(i < j) == (j < k)
```

Logical Operators 논리 연산자

- 좀 더 복잡한 논리 표현식은 ***logical operators 논리 연산자로:***

!	logical negation	(unary)
&&	logical *and*	(binary)
||	logical *or*	(binary)

결과
- 0 은 false
- 1 은 true

피 연산자
- 0 은 false으로 처리
- > 0 은 true로 처리

- 논리 연산자의 동작:

  !*expr* 의 결과는 *expr* 의 값이 0 일 때만 참. 참이면 거짓, 거짓이면 참

  expr1* && *expr2 는 *expr1* 와 *expr2*의 값이 0이 아닐 때만 참.
  둘 다 참이면 참, 아니면 거짓

  expr1* || *expr2 의 결과는 *expr1* 와 *expr2* 중 하나 또는 둘 다 0이 아닐 때만 참. 둘 중 하나가 참이면 참, 둘 다 거짓이면 거짓

Logical Operators

- `&&` 와 `||` 는 빠른 판단이 가능함. 왼쪽 먼저 검사 후 판단 가능하면 오른쪽 검사 안함
- Example:

```
(i != 0) && (j / i > 0) // 영으로 나누기 방지
```

Logical Operators

- `!` 연산자는 단항 연산자와 같은 우선순위를 갖음

- `&&` 와 `||` 의 우선 순위는 관계연산자와 동치 연산자보다 낮음
  - 예, `i < j && k == m` 는 `(i < j) && (k == m)` 을 뜻함.

- `!` 연산자는 오른쪽 결합이고; `&&` 와 `||` 는 왼쪽 결합 임

if 문

- If문은 표현식 검사 결과에 따라 서로 다른 문장을 선택할 수 있도록 함
- 가장 간단한 형태의 if 문은 다음과 같음

  if (*expression*) *statement*

- if 문이 실행되어, 표현식 *expression* 이 평가되어 0이 아닌 값이 나오면 *statement* 문장이 실행됨
- Example:

```
if (line_num == MAX_LINES)
  line_num = 0;
```

The `if` Statement

- `==` (equality 동치) 를 `=` (assignment 할당)과 혼돈하는 것이 **가장 흔한 실수**
- 다음의 문장은

```
if (i == 0) …
```

`i` 가 0 인지 비교함.

- 다음의 문장은

```
if (i = 0) …
```

`i`에 `0`을 할당하고, 0이 아닌지 판단하는 문장임

The `if` Statement

- 많은 경우 `if`문에 비교문장은 어떤 특정 값의 범위 내에 있는 검사하기 위해 사용됨
- $0 \le$ `i` < `n`인지 검사:

```
if (0 <= i && i < n) …
```

- 그 반대의 경우 검사 (`i` 가 범위 밖에 존재):

```
if (i < 0 || i >= n) …
```

Compound Statements 합성 문장

- `if` 문의 형식에서 *statement는 단수로 표현됨*:

  `if` `(` *expression* `)` *statement*

- `if` 문이 두 개 이상의 문장을 제어하기 위해서 ***compound statement 합성 문장을 써야 함.***
- 합성 문장은 다음의 형식을 갖음

  `{` *statements* `}`

- 여러 문장을 중괄호로 묶으면 컴파일러가 보기에는 마치 한 묶음으로 처리해야 하는 문장으로 보게 됨

Compound Statements

- Example:

```
{ line_num = 0; page_num++; }
```

- 합성 문장은 여러 줄로 나뉘어 작성됨. 한 문장이 한 줄

```
{
  line_num = 0;
  page_num++;
}
```

- `if` 문자에 쓰인 합성 문자의 예:

```
if (line_num == MAX_LINES) {
  line_num = 0;
  page_num++;
}
```

else Clause (절)

- If 문은 else 절이 있을 수 있음:

  if (*expression*) *statement* else *statement*

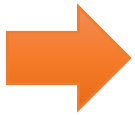
- else 뒤에 따르는 문장은 검사식이 0일 때 수행됨
- Example:

```
if (i > j)
  max = i;
else
  max = j;
```

The `else` Clause

- `if` 문은 또 다른 `if` 문 내에 포함 될 수 있음: 중첩가능

```
if (i > j)
  if (i > k)
    max = i;
  else
    max = k;
else
  if (j > k)
    max = j;
  else
    max = k;
```



```
if (i > j) {
  if (i > k)
    max = i;
  else
    max = k;
} else {
  if (j > k)
    max = j;
  else
    max = k;
}
```

```
if (i > j) {
  if (i > k) {
    max = i;
  } else {
    max = k;
  }
} else {
  if (j > k) {
    max = j;
  } else {
    max = k;
  }
}
```

- 각`else` 절을 `if` 문과 정렬하면 읽기 쉬워짐

중첩 `if` 문

- "cascaded 중첩" `if` 문은 연달은 검사 조건을 평가하기에 유용함

- Example:

```
if (n < 0)
  printf("n is less than 0\n");
else
  if (n == 0)
    printf("n is equal to 0\n");
  else
    printf("n is greater than 0\n");
```

Cascaded `if` Statements

- 두 번째 `if` 문이 첫 번째 `if` 내에 중첩되었지만 분리해서 쓰지는 않음

- 대신 `else` 를 최초의 `if`에 맞춰서 작성함:

```
if (n < 0)
  printf("n is less than 0\n");
else if (n == 0)
  printf("n is equal to 0\n");
else
  printf("n is greater than 0\n");
```

Cascaded `if` Statements

- 이런 식으로 표현하면 들여쓰기를 너무 많이 해야 하는 경우를 방지할 수 있음:

```
if ( expression )
   statement
else if ( expression )
   statement
…
else if ( expression )
   statement
else
   statement
```

The "Dangling `else`" Problem

- `If`이 중첩되면 "매달린 `else`" 문제가 생길 수 있음:

```
if (y != 0)
  if (x != 0)
    result = x / y;
else
  printf("Error: y is equal to 0\n");
```

- 들여쓰기 상으로는 `else` 절이 처음의 `if` 문에 연결된 것처럼 보임.
- 하지만 C의 규칙은 `else` 절은 가장 가까운 `if` 문에 연결됨

Conditional Expressions 조건식

- C의 ***conditional operator*** 조건 연산자는 조건 계산 결과에 따라 두 개 중 하나의 값을 표현함
- 조건 연산자는 두개의 기호로 구성 (? 와 :), 항상 같이 쓰임:

  expr1 ? expr2 : expr3
- 피연산자의 형은 상관 없음.
- 이런 표현식을 조건식 ***conditional expression***이라 부름.
- 삼항 ***ternary*** 연산자라고도 함.
- 조건식 *expr1 ? expr2 : expr3* 은"만약 *expr1* 이 참이면 *expr2* 아니면 *expr3*"라고 읽음

Conditional Expressions

- Example:

```
int i, j, k;

i = 1;
j = 2;
k = i > j ? i : j;              /* k is now 2 */
k = (i >= 0 ? i : 0) + j;       /* k is now 3 */
```

짧지만 난해함

- 괄호가 꼭 필요함. 조건식의 기호들은 앞에 말한 연산자들보다 우선 순위가 낮기 때문

- 조건식은 `return` 문에 주로 쓰임:

```
return i > j ? i : j;
```

Conditional Expressions

- `printf` 호출에 조건식을 쓰면 유용함

```
if (i > j)
  printf("%d\n", i);
else
  printf("%d\n", j);
```

처럼 쓰는 대신

```
printf("%d\n", i > j ? i : j);
```

- 조건식은 매크로 정의에 많이 쓰임

Boolean 값: 1 또는 0; true 또는 false

Old way

```
int flag;

flag = 0;
…
flag = 1;
```

이해하기 어려움

```
if (flag == TRUE)
 …
if (flag == FALSE)
 …
```

C89

```
#define TRUE  1
#define FALSE 0

flag = FALSE;
…
flag = TRUE;
```

이해가 자연스러움

C89 Better Usage

```
#define BOOL int

BOOL flag;
```

불리언 조건이 명확함

```
if (flag)
 …
if (!flag)
 …
```

C99 표준에서 불리언 값

- C99 는 `_Bool` 형을 제공함.
- 불리언 변수는 다음과 같이 선언됨

```
_Bool flag; //특별한 변수 형으로 0 과 1만 있음
flag = 5;   /* flag에 1이 할당됨 */
```

- C99의 `<stdbool.h>` 헤더에는 `_Bool`를 뜻하는 `bool`을 정의하고 있음.

```
#include <stdbool.h>
 bool flag;   /* same as _Bool flag; */
```

- 그리고 1 과 0을 뜻하는 `true` 와 `false` 매크로도 존재함

```
flag = false;
flag = true;
```

switch 문

중첩 if 문

```
if (grade == 4)
  printf("Excellent");
else if (grade == 3)
  printf("Good");
else if (grade == 2)
  printf("Average");
else if (grade == 1)
  printf("Poor");
else if (grade == 0)
  printf("Failing");
else
  printf("Illegal grade");
```

switch 문

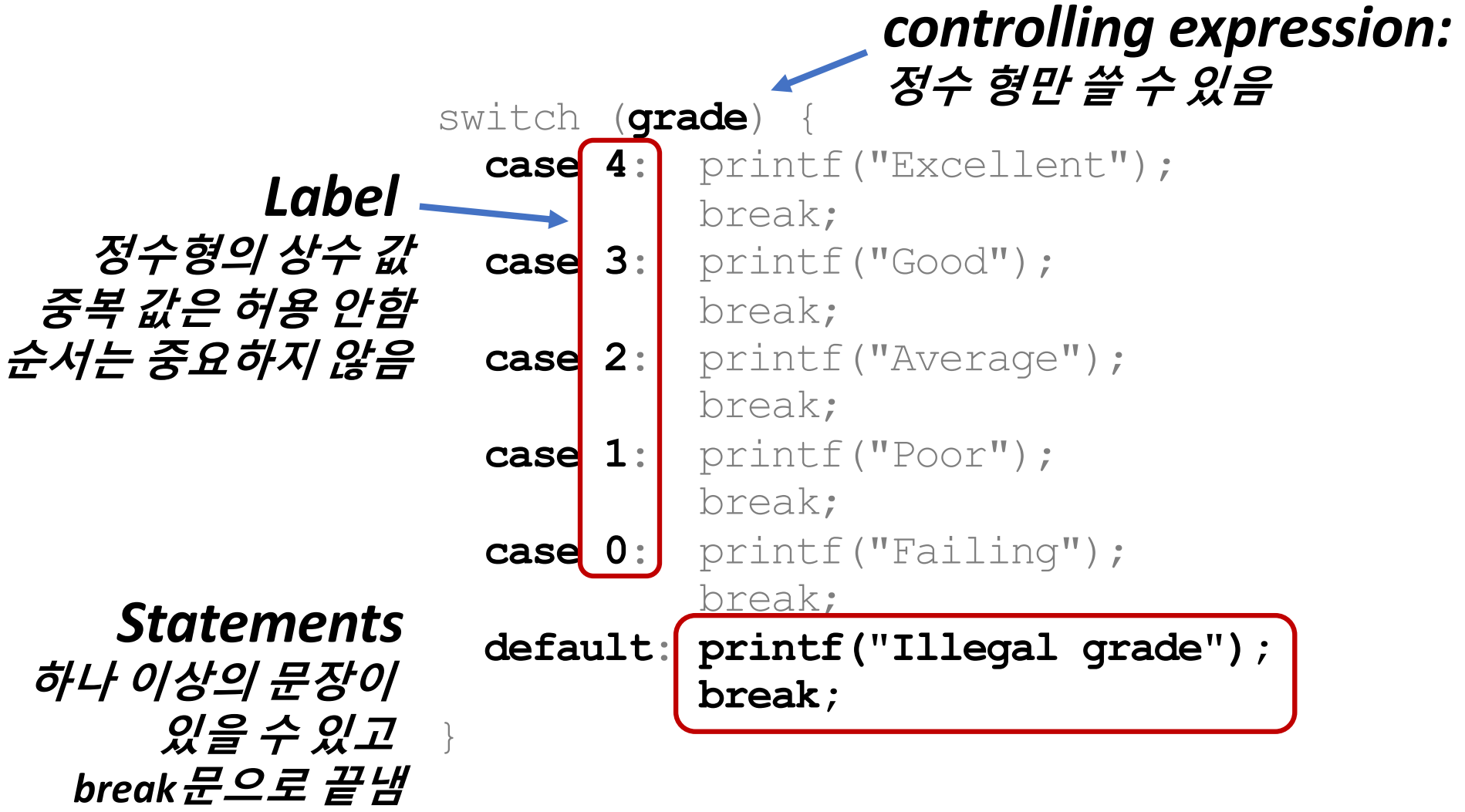
```
switch (grade) {
  case 4:  printf("Excellent");
           break;
  case 3:  printf("Good");
           break;
  case 2:  printf("Average");
           break;
  case 1:  printf("Poor");
           break;
  case 0:  printf("Failing");
           break;
  default: printf("Illegal grade");
           break;
}
```

The `switch` Statement

- `switch` 문은 중첩 `if` 문 보다 읽기 쉬움.
- `switch` 는 보통 `if` 보다 빠름.
- `switch` 문의 형태:

```
switch ( expression ) {
  case constant-expression : statements
  …
  case constant-expression : statements
  default : statements
}
```

The `switch` Statement



The `switch` Statement

- 어떤 문장 전에는 여러 라벨이 있을 수 있음:

```
switch (grade) {
  case 4:
  case 3:
  case 2:
  case 1:  printf("Passing");
           break;
  case 0:  printf("Failing");
           break;
  default: printf("Illegal grade");
           break;
}
```

- `default` 케이스가 없고, 제어식에 따라 맞는 케이스가 없으면 `switch` 다음의 문장을 실행함