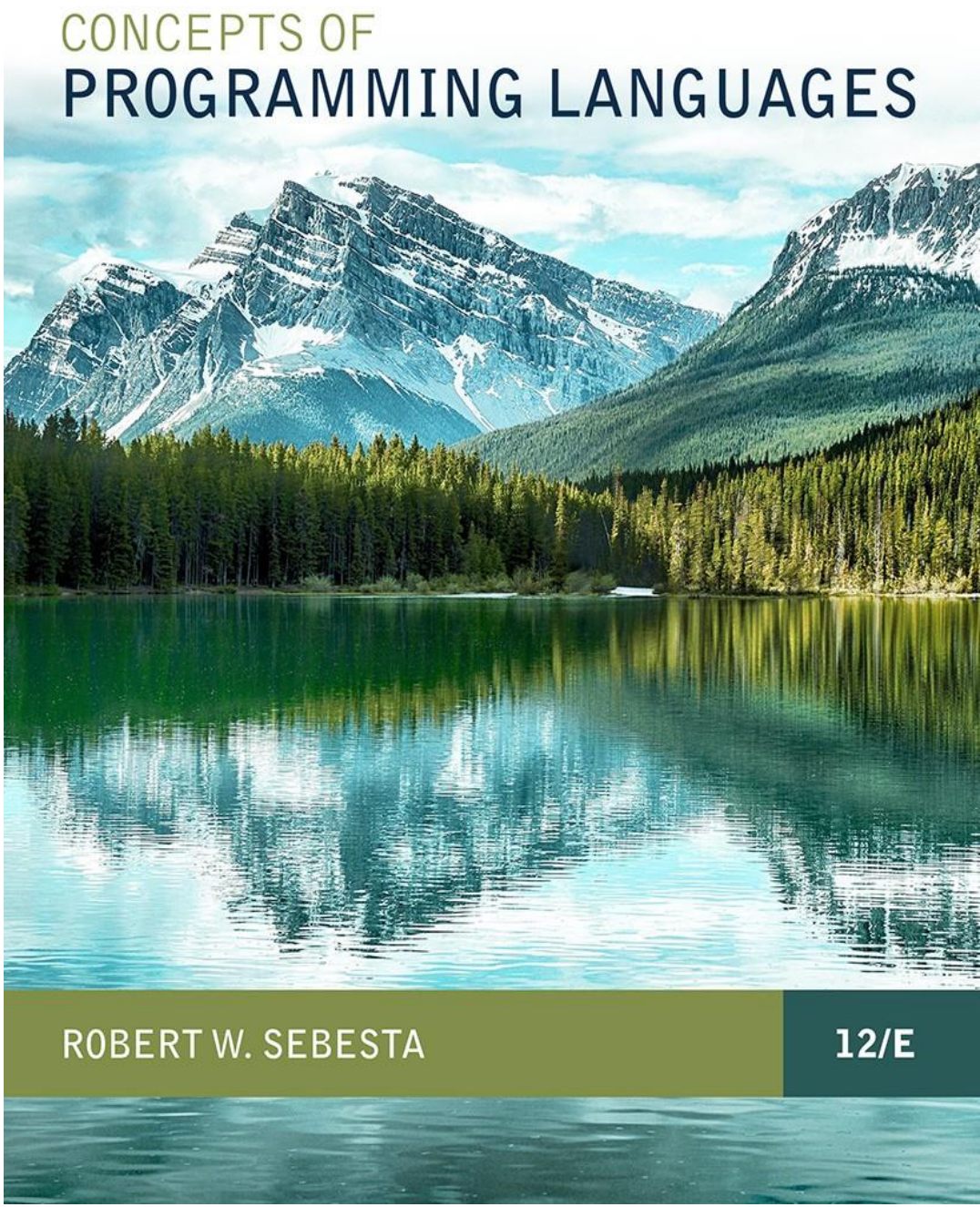
Chapter 8

Statement-Level Control Structures



ISBN 0-321-49362-1

Chapter 8 Topics

- 서론
- 선택문
- 반복문
- 무조건 분기

제어 흐름의 수준

- 식 내부에서 (Chapter 7)
- 프로그램 단위 간 (Chapter 9)
- 프로그램 문장들 간 (this chapter)

제어 구조

- 제어 구조(*control structure*)는 제어문과 제어문이 실행을 제어하는 문장들의 집합

- 설계 고려사항
  - 제어 구조가 여러 개의 진입점을 가져야 하는가?
    - 다중진입은 goto와 문장레이블을 제공하는 언어만 가능
    - 비교: 다중출구는 대부분 허용

선택문

- 선택문은 두 개 혹은 그 이상의 실행 경로들 중 하나를 선택하는 수단

- 두 가지 일반적인 범주 :
  - 2 방향 선택자
  - 다중 선택자

2 방향 선택문

- 일반 형식: **`if`** 제어식
  `then` 절
  `else` 절
- 제어식
  - if (제어식) 절
  - if 제어식 then 절

  - C89, C99, Python, C++ : 산술식이 제어식으로도 사용 가능
  - 대부분 다른 언어 : 불리안 식만이 제어식으로 사용

절 형식

- 많은 언어에서, **then**과 **else** 절은 단일문/복합문
- **Perl** : 모든 절은 중괄호를 사용하여 구분 (한 개의 문장인 경우에도 복합문이어야 함)
- **Python, Ruby** : 절은 일련의 문장들임
- **Python**은 들여쓰기를 사용하여 절을 명세함

```
if x > y :
  x = y
  print "x was greater than y"
```

선택자 중첩

- Java example

```
if (sum == 0)
  if (count == 0)
      result = 0;
 else result = 1;
```

- Which **`if`** gets the **`else`**?
- Java's static semantics rule : **`else`** matches with the nearest previous **`if`**

선택자 중첩 (continued)

- Java에서, 첫 번째 then절에 대한 양자택일의 의미를 주려면 내부 if를 복합문으로 표현

```
if (sum == 0) {
 if (count == 0)
      result = 0;
}
else result = 1;
```

- C, C++, C#에서도 동일

선택자 중첩 (continued)

- 선택구조의 끝을 표시하는 특수어(end) 사용 : Ruby

```
if sum == 0 then
  if count == 0 then
    result = 0
  else
    result = 1
  end
end
```

```
if sum == 0 then
  if count == 0 then
    result = 0
  end
else
  result = 1
end
```

선택자 중첩 (continued)

- Python

```
if sum == 0 :
    if count == 0 :
        result = 0
    else :
        result = 1
```

선택자 식

- 함수형 언어 ML, F#, LISP에서, 선택자는 문장이 아니라 값을 반환하는 식이다.
- F#의 예

```
let y =
    if x > 0 then x
    else 2 * x
```

※ 만약 **if** 식이 값을 반환하면 **else** 절을 가져야 함
반환하는 값의 타입은 동일해야 함

다중 선택문 : Examples

- C, C++, Java, JavaScript

```
switch (expression) {
  case const_expr1: stmt1;
  ...
  case const_exprn: stmtn;
  [default: stmtn+1]
}
```

다중 선택문 : Examples

- C의 **switch문에서** 설계 선택사항
  1. 제어식과 상수식은 정수 타입(문자, 열거타입)이어야 함
  2. 선택가능한 세그먼트는 statement sequences, blocks, or compound statements
  3. 하나의 선택구조 실행시 여러 세그먼트가 실행될 수 있음 (세그먼트의 끝에서 묵시적인 분기가 없음)
  4. Default 절은 표현되지 않은 값을 위한 절임 (if there is no **default**, the whole statement does nothing)

다중 선택문 : Examples

- C#
  - 하나 이상의 세그먼트의 암묵적 실행을 허용하지 않는 정적 의미 규칙을 갖고 있다는 점에서 C와 다름

  1) 각 세그먼트는 명시적 무조건 분기문(**goto** or **break**)으로 끝나야 함(예제)

  2) 제어식과 **case** 상수가 문자열 가능

다중 선택문 : Examples

- C#의 예제

```
switch (value) {
    case -1:
        Negatives++;
        break;
    case  0:
        zeros++;
        goto case 1;
    case  1:
        Positives++;
        break;
    default :
        Console.WriteLine("Error inswitch  \n");
}
```

다중 선택문 : Examples

- Ruby 는 case 문장에 두 가지 형식을 지원
  1) 중첩 if문 리스트와 의미 유사

```
leap = case
   when year % 400 == 0 then true
   when year % 100 == 0 then false
   else year % 4 == 0
   end
```

  2) Java의 switch와 유사

- Perl, Python은 다중 선택문 없음

다중 선택문의 구현

다중 조건 분기 명령어

```
switch(식) {
  case 상수식1: 문장1;
        break;
   . . .
 case 상수식n: 문장n;
        break;
[default : 문장n+1]
```

→

```
식을 t로 평가하는 코드
goto branches
레이블1:  문장1에 대한 코드
            goto out
   . . .
레이블n:  문장n에 대한 코드
            goto out
default : 문장n+1에 대한 코드
          goto out
banches: if  t=상수식1  goto 레이블1
          . . .
          if  t=상수식n  goto 레이블n
          goto default
out:
```

If를 이용한 다중 선택

- else-if 절을 사용하여 2-방향 선택자를 다중 선택자의 확장으로 나타낼 수 있음.

  예를 들어, Python에서

```
if count < 10 :
  bag1 = True
elif count < 100 :
  bag2 = True
elif count < 1000 :
  bag3 = True
```

If를 이용한 다중 선택

- Python 예제는 Ruby case 로 작성될 수 있음

```
case
  when count < 10 then bag1 = true
  when count < 100 then bag2 = true
  when count < 1000 then bag3 = true
end
```

Scheme의 다중 선택문

- COND라는 특정형식의 함수
- 함수의 일반형:

```
(COND
  (술어1 식1)
   …
  (술어n 식n)
  [(ELSE 식n+1)]
 )
```

  - ELSE 절은 선택사항; ELSE 는 **true**의 동의어
  - 각 (술어-식) 쌍은 매개변수
  - 의미 : 첫 번째 쌍부터 술어를 평가하여 최초로 **true**인 쌍의 식을 평가한 값이 **COND**의 값으로 반환됨

Scheme의 다중 선택(예제)

```
(COND
  ((> x y) "x is greater than y")
  ((< x y) "y is greater than x")
  (ELSE "x and y are equal")
 )
```

반복문

- 하나의 문장이나 복합문의 반복 실행 : **by iteration or recursion**
- 반복 제어문의 일반적인 설계 고려사항 :
  1. 반복이 어떻게 제어되는가?
  2. 제어 매커니즘이 루프의 어디에 위치해야 하는가?

계수기-제어 루프(Counter-Controlled Loops)

- 계수 반복 제어문은 루프 변수를 가지며, 이 변수에 계수 값이 유지된다. (*initial, terminal*, and *stepsize* values를 명세하는 수단이 됨)
- 설계 고려사항 :
  1. 루프 변수의 타입과 영역은?
  2. 루프 내부에서 루프 변수나 루프 매개변수가 변경되는 것이 적법한가? 그렇다면 그 변경이 루프 제어에 영향을 미치는가?
  3. 루프 매개변수는 단지 한번만 평가되는가? 아니면 매 반복마다 한번씩 평가되는가?

계수기-제어 루프 : Examples

- C-기반 언어

  for ([식1] ; [식2] ; [식3])
  루프몸체

  - 루프몸체는 단일문, 복합문, 콤마로 구분되는 문장, 널 문장
  - 식2가 생략되면 무한 루프
  - 연산의미론

```
   식1
loop:
   if 식2=0 goto out
   for var=[이산범위의 다음 번째 원소]
   [루프 몸체]
   식3
   goto loop
out:  ...
```

계수기-제어 루프 : Examples

- C++ 은 두 가지 면에서 C 와 다름:
  1. 제어 식이 Boolean일 수도 있음
  2. 초기식은 변수 정의를 포함할 수 있음 (영역은 정의로부터 루프 몸체 끝까지임)
- Java and C#
  - 제어식이 Boolean이어야 한다는 점에서 C++과 다름

계수기-제어 루프 : Examples

- Python

  for 루프_변수 **in** 객체:
  - 루프 몸체

  [**else**:
  - else 절]

```
for count in [2, 4, 6]:
    print count
```

- 객체는 보통 범위 임, brackets (`[2, 4, 6]`)처럼 값의 리스트이거나 range 함수호출 (**range**(5)의 호출은 0, 1, 2, 3, 4를 반환

- 루프 변수는 주어진 range에 명세한 값을 매 반복마다 하나씩 받아들임

- 선택적인 else절은 만약 루프가 정상으로 끝난다면 실행될 수 있음

계수기-제어 루프 : Examples

- F#
  - 계수기(counters)는 변수가 필요하나 함수형 언어는 변수를 갖지 않으므로 계수기-제어 루프는 재귀함수로 반복을 제어한다.

```
let rec forLoop loopBody reps =
  if reps <= 0 then ()
  else
    loopBody()
    forLoop loopBody, (reps - 1);;
```

  - 매개변수 `loopBody` (루프의 몸체를 정의하는 함수) 와 반복횟수 `reps`를 가진 재귀함수 `forLoop` 의 정의
  - `()` means do nothing and return nothing

논리-제어 루프(Logically-Controlled Loops)

- 반복 제어는 **Boolean** 식에 기반한다.
- 설계 고려사항 :
  - **Pretest or posttest?**
  - 논리-제어 루프가 계수 루프의 특별한 형식이어야 하는가, 아니면 별도의 문장이어야 하는가?

논리-제어 루프 : Examples

- C 와 C++은 pretest와 posttest 형식 모두를 가짐;
  여기서 제어식은 산술식도 가능

while (제어식)
    loop body

do
    loop body
while (제어식)

  - C 와 C++에서 논리-제어 루프의 몸체로 분기하는 것이 적법하다.

- Java는 C 와 C++과 유사하나, 다른 점은 제어식이
  Boolean 타입이어야 하며 루프 몸체는 그 시작
  위치에서만 진입될 수 있다 (-- Java는 **goto가 없음**)

사용자-지정 루프 제어 매커니즘
(User-Located Loop Control Mechanisms)

- 가끔 프로그래머가 루프제어의 위치(처음이나 끝이 아닌)를 결정하는 것이 편리
- 단일 루프에 대해서 설계는 간단(e.g., **`break`**)
- 중첩 루프에 대한 설계 고려사항
  1. 조건 매커니즘이 탈출(exit)의 필수적인 부분인가?
  2. 제어가 한 개 이상의 루프로부터 탈출 가능한가?

사용자-지정 루프 제어 매커니즘 (continued)

- C , C++, Python, Ruby, C# 은 무조건이면서 레이블이 없는 탈출(**break**)을 제공
- Java 와 Perl 은 무조건이면서 레이블을 갖는 탈출 (**break** in Java, **last** in Perl)을 제공
- C, C++, Python 은 레이블이 없는 **continue**를 포함; 이는 현재 반복의 나머지를 건너 뛰지만 루프는 계속됨
- Java 와 Perl 은 **continue**의 레이블이 있는 버젼을 가짐 (예제 : p399 참조)

- 중첩루프로부터 외곽루프로 탈출하는 예(Java의 break)

```
outerLoop:
  for (row=0; row<numRows; row++)
     for (col=0; col<numCols; col++)
        {  sum += mat[row][col];
           if (sum>1000.0)
              break outerLoop;
        }
```

- 가장 작은 중첩루프로 제어를 전달하는 예(C, C++, Python의 continue)

```
while (sum < 1000) {
        getnext(value);
        if (value < 0)  continue;        //       break;  ??
        sum += value;
     }
```

데이터 구조에 기반한 반복

- 데이터 구조에 있는 원소의 수가 루프 반복을 제어
- 데이터-기반 반복문은 사용자정의 데이터 구조와 사용자정의 함수(반복자)를 사용하여 그 구조의 원소들에 대해 반복함

  (ex) (Python) for count in range [0, 9, 2]:
- 제어 매커니즘은 반복자(iterator) 함수를 호출하여 어떤 특정 순서로 다음 원소를 반환하게 함
- C의 **for**를 사용하여 사용자-정의 반복자를 구축 가능

  (ex) 이진트리의 노드들을 처리한다면

```
for (p=root; p!=NULL; traverse(p)){
   ...
}
```

데이터 구조에 기반한 반복 (continued)

- C# and F# (and the other .NET languages)
  - 포괄적 라이브러리 클래스를 가짐
    (for arrays, lists, stacks, and queues).
  - **foreach** 문과 함께 묵시적인 내장 반복자를 포함

(C#의 예)

```
List<String> names = new List<String>();
names.Add("Bob");
names.Add("Carol");
names.Add("Ted");
      . . .
foreach (Strings name in names)
    Console.WriteLine (name);
```

무조건 분기

- 실행 제어를 프로그램 상의 명세된 위치로 이동
- 1960년대와 1970년대 가장 열띤 토론 주제
- 주요 관심사 : Readability
- 몇몇 언어들은 **goto** 문을 지원하지 않음 (e.g., Java)
- C# 은 **goto** 문 지원 (**switch** 문에서만 사용가능)
- 루프 탈출문은 일종의 제한된 **goto** 임; 판독성에 해가 되지 않음