

프로그래밍 언어론

반복문(Iterative Statement)

컴퓨터공학과
조은선



충남대학교
CHUNGNAM NATIONAL UNIVERSITY

반복문 (Iterative statement)

학습목표

프로그래밍 언어의 구문요소 중에서 문장 실행 흐름을 제어하는 control structure 중 반복문(iterative statement)의 의미를 공부한다.

학습내용

- 각종 반복문의 종류
- 무조건 분기



목 차

- 들어가기
- 학습하기
 - 분기문 개요
 - 횟수 제어(Counter-controlled) loop 예) for 문
 - 논리 제어(Logically-controlled) loop : 예) while 문
 - 데이터 구조에 기반한 반복
 - 반복연산자 (iterator)
 - 사용자 지정 loop control (goto)
- 평가하기
- 정리하기



알고가기



다음 C 코드 중 가장 `printf`의 반복횟수가 가장 많은 것은?

- 1) `for (i = 0; i < 10 ; i++)
 printf("a");`
- 2) `i = 1; while (i <= 10) {
 i++;
 printf("a");
}`
- 3) `i = 1; while (1) {
 printf("a");
 if (i > 10) break;
 i++;
}`
- 4) `for (i=10; i>0; i--)
 printf("a");`



반복문(Iterative Statement)

반복된 실행

- 반복: 문장 수준 제어
- 재귀: 단위 프로그램 수준 제어

반복 제어문을 위한 설계 쟁점

- 반복의 제어는 어떻게 되는가?
- loop의 제어 매커니즘은 어디에 있는가

두 설계 쟁점에 대한 공통된 전략

- Counter-controlled(횟수 제어) loop 예) for 문
- Logically-controlled(논리 제어) loop 예) while 문
- 그 밖에

데이터 구조에 기반한 반복, 반복자 (iterator), 사용자 지정 loop control, goto ...



Counter Controlled(횟수 제어) Loop

설계 쟁점

- >> loop 변수의 타입과 범위는 무엇인가?
- >> loop 종료시 loop변수의 값은 무엇인가?
- >> loop 의 body 에서 loop 변수 혹은 loop 매개변수의 수정이 허용되는가? 만약 그렇다면 loop 제어에 영향을 주는가?



C, Java, C++ 등의 Counter-Controlled Loop

C

```
for ([expr_1] ; [expr_2] ; [expr_3]) statement
```

- ⇒ `expr_k`는 모든 문장이 될 수 있음, 혹은 콤마에 의해 분리되는 문장 나열이 될 수도 있음
- ⇒ 문장 나열의 결과값은 마지막 문장의 값
- ⇒ 만약 `expr_2` (loop control expression) 가 없다면, 무한loop임

C++

- ⇒ C와 비슷하나, 제어식 (`expr_2`) 은 boolean 식이 될 수도 있고, 초기식 (`expr_1`) 에 변수 정의를 포함할 수 있음 (변수의 범위는 정의부터 loop body까지임)

Java

- ⇒ C와 비슷하나 제어식 (`expr_2`)는 반드시 boolean 식임



기타 언어들의 Counter-Controlled loop

C 언어

```
for (i=0; i < 10 ; i+=2) stmt
```

Fortran86

```
Do i=0, 8, 2  
stmt  
END DO
```

Algol60

```
for index := 0 step 2 until 8
```

Perl

```
foreach $i (0, 2, 4, 6, 8)      stmt
```

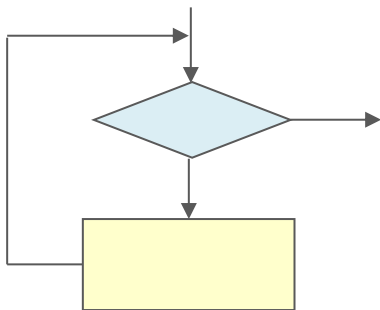


Logically Controlled (논리 제어) loop

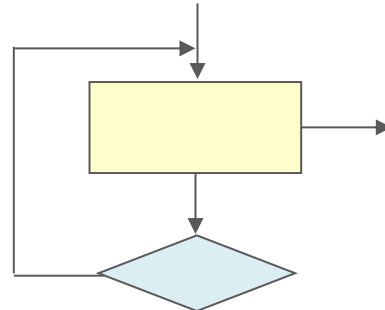
설계 쟁점

사전 검사 혹은 사후 검사?

이는 횟수 loop문의 특별한 경우인가?
별도의 구문인가?



pre-test loop



post-test loop



Logically Controlled loop: 언어들의 예

C와 C++은 별도의 사전검사 (while, true-test)와
사후검사 (do-while, true-test) 논리 loop가 있음

Java는 다음을 제외하고 C 유사,

- 제어식은 반드시 부울식이어야 함
- 몸체는 반드시 처음에서만 들어 갈 수 있음 (Java는 goto가 없음)

Ada, Perl 는 사전검사는 있지만, 사후검사는 없다

Fortran 77과 90는 아예 없음



데이터 구조에 기반한 반복

개 념

- 반복 제어를 위해 순서가 있는 일정한 수의 데이터 구조 원소를 사용하는 제어 메커니즘
- 선택된 순서화된 집합에서
만약 다음 원소가 있다면 다음 원소를 리턴하는 iterator 함수를 호출하고,
그렇지 않으면 loop를 탈출

예제

• C#

```
String() strList = {"Bob", "Carol", "Ted"};  
foreach (String name in strList) { ... name ... }
```

• C

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

iterator function



Iterator(반복연산자)

개 념

- ➔ 순서가 있는 데이터 구조의 모든 노드를 방문해서
- ➔ 각 노드에 사용자가 제공하는 연산을 수행

•수동 iterator

- ➔ 사용자가 노드를 처리하는 메소드를 해당 객체에 제공. 객체는 모든 노드에 순서대로 메소드를 적용

예제) Smalltalk

```
class List {
    Object[] listElements = new Object[size];
    ...
    public void do( Function userOperation ) {
        for (int k=0; k < listElements.length(); k++)
            userOperation( listElements[k] );
    }
}

List grades = new List();
aFunction = (item){ print(item) }; // 수행할 메소드
...
grades.do( aFunction );           // 메소드 전달
```



능동 iterator

•능동 iterator

➡ 사용자가 객체의 다음 원소로 이동하고, 처리하는 것을 책임짐.

➡ 예제) Java, C++

```
List grades = new List();  
  
Iterator gradeList = grades.iterator();  
  
while ( gradeList.hasNext() ) {  
    listItem = gradeList.next();    // 다음 원소로 이동  
    print ( listItem );            // 원소의 처리  
}
```



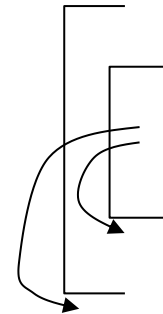
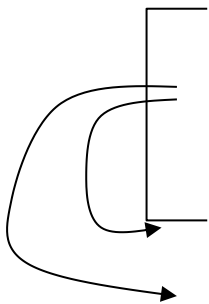
사용자 지정 loop 제어

“제어를 loop 밖으로 옮기는 것”

설계 쟁점

loop 탈출에 조건부 매커니즘이
통합되어 있나?

제어는 하나 이상의 중첩 loop를
벗어날 수 있어야 하는가?



사용자 지정 loop 제어 : Ada

- ➔ 조건부 혹은 무조건으로; 어떠한 loop에서도; 어떠한 수의 레벨에서도 제어가 loop 밖으로 나갈 수 있음

```

for ... loop
    ...
    exit when ...
    ...
end loop

```

```

LOOP1:
  while ... loop
    ...
    LOOP2:
      for ... loop
        ...
        exit LOOP1 when ...
        ...
      end loop LOOP2;
    ...
  end loop LOOP1;

```



사용자 지정 loop 제어 : C, C++, Java

➔ break

- 무조건 탈출; 어떠한 loop나 **switch**에서;
하나의 레벨만 탈출 가능

```
break;
```

➔ continue

- 반복의 나머지 부분을 건너뛰지만, loop를 탈출하지는 않음

```
continue;
```



무조건 분기(goto)

→ 문제점: 가독성

어떤 언어는 가지고 있지 않음: Java

→ loop 탈출문은 모두 본질적으로 goto임

`continue, break, exit, last ...`

→ 일종의 문장 레이블을 필요로 함

예) PL/I: 레이블이 변수처럼 사용됨 (변수 타입이 레이블)

- 값이 할당되거나 매개변수로 전달할 수 있음
- 매우 유연하지만, 프로그램을 거의 읽을 수 없게 하고 구현하기 어렵게 함

→ goto 의 목적지는 다음 중 하나가 되어야 함

1. goto 를 포함하고 있는 제어문 또는, goto 문장이 포함되어 있는 statement 그룹 내에 있는 문장
2. goto 를 포함하는 statement 들을 포함하는 statement 그룹 내에 있는 문장
3. goto 가 있는 서브프로그램을 둘러싸고 있는 서브프로그램의 범위에 있는 문장으로서, 문장그룹의 내부가 아닌 문장
 - goto 가 어떠한 수의 서브프로그램도 종료시킬 수 있음을 의미

4. 하지만, 절대로 같은 레벨이나 goto보다 더 안쪽으로 nested 된 statement 그룹에는 goto의 목적지가 있을 수 없다.



평가하기

1 다음 중 반복문에 대한 설명으로 틀린 것은?

- ① 능동 iterator는 사용자가 객체의 다음 원소로 이동하고, 처리하는 것을 책임짐.
- ② 모든 logically controlled loop는 counter controlled loop로 표현할 수 있다.
- ③ loop 탈출문은 본질적으로 무조건 분기이다.
- ④ C와 C++은 별도의 사전검사 loop와 사후검사 loop가 모두 존재한다.

확인



평가하기

2 다음 중 Perl 구문에 대한 설명으로 틀린 것은?

보기 `foreach $i (0, 2, 4, 6, 8) stmt`

- ① Logically controlled Loop 이다.
- ② `$i`는 loop 매개변수이다.
- ③ `$i`가 0, 2, 4, 6, 8인 경우에만 `stmt`가 수행된다.
- ④ C 문장 `for (i = 0; i < 10; i+=2) stmt` 과 동일하다.

확인



정리하기

- ➡ counter-controlled loop와 logically controlled loop의 다양한 변형들이 사용됨
- ➡ iterator 함수를 통한 데이터 구조의 탐색 처리가 널리 사용됨
- ➡ 무조건 분기 goto는 현대의 언어에서는 권장하지 않음

“모든 프로그램의 control structure는 양자 선택문과 사전검사 logically controlled loop면 충분히 표현된다. 그러나 작성 편의를 위해 다른 control structure들도 존재한다.”

